

ipd4600magridpmTES-10

**Defense Information Infrastructure (DII)
Common Operating Environment (COE)**

**Programming Manual (PM)
for the
Grid Field API (MAGRID) Segment
of the
Tactical Environmental Support System Next Century
[TESS(NC)]
Meteorology and Oceanography (METOC) Database**

Document Version 4.6

29 January 1999

**Prepared for:
Naval Research Laboratory
Marine Meteorology Division
Monterey, CA**

**Prepared by:
Integrated Performance Decisions
Middletown, RI**

PRINTED COPY IS UNCONTROLLED AND MAY BE OBSOLETE

ipd4600magridpmTES-10

Table of Contents

1	SCOPE.....	1
1.1	Identification	1
2	REFERENCED DOCUMENTS	5
2.1	Government Documents	5
2.2	Non-Government Documents.....	6
3	SEGMENT OVERVIEW.....	7
4	SEGMENT DEVELOPMENT	9
4.1	Writing Applications Using the MAGRID APIs.....	11
4.1.1	Ingesting a 2D Grid Field Into the Database.....	11
4.1.2	Retrieving Center, Model, Parameter, and Unit Information from the Database	13
4.1.3	Ingesting a 3D Grid Field Into the Database.....	18
4.1.4	Registering Grid Models Into the Database	22
4.1.5	Deleting Single Grid Fields From the Database	24
4.1.6	Deleting Multiple Grids From a 2D or 3D Dataset.....	27
4.1.7	Retrieving 2D Grid Fields From the Database.....	28
4.1.8	Retrieving 3D Grid Field Data From the Database.....	32
4.1.9	Ingesting/Retrieving a 3D EOF.....	42
4.1.10	Updating Grid Field Points in the Database.....	46
4.1.11	Getting a 2DCatalog of Grid Fields From the Database	49
4.1.12	Getting a 3DCatalog of Grid Fields From the Database	49
4.1.13	Retrieving Registration(s) From the Database	50
4.1.14	Getting a Single Point From a Given 2D Grid Field.....	53
4.2	MDGRID Database Reference Tables	56
4.2.1	MDGRID Units Table.....	56
4.2.2	MDGRID Geophysical Parameters Table	62
4.2.3	MDGRID Site-Specific Geophysical Parameters Table	67
4.2.4	MDGRID Production Centers Table.....	74
4.2.5	MDGRID Models Table	76
4.2.6	MDGRID Registration Table	77
4.2.7	MDGRID Area of Interest Table	78
4.3	Building Applications Using the MAGRID Libraries.....	79
4.3.1	Makefile Using the Dynamic/Static MAGRID Libraries on HP-UX	79
4.3.2	Makefile Using the Dynamic MAGRID Libraries on Windows NT	80
4.3.3	Makefile Using the Static MAGRID Libraries on Windows NT	81
5	CUSTOMIZING SEGMENTS	83

6	NOTES	85
6.1	Glossary of Acronyms.....	85
Appendix A - Scan Mode Values.....		A-1

List of Figures

1-1	TESS(NC) METOC Database Conceptual Organization	3
------------	--	----------

1 SCOPE

1.1 Identification

This Programming Manual (PM) describes the use of the Grid Field Application Program Interface (API) (MAGRID) segment, Version 4.3.0.0, of the Tactical Environmental Support System Next Century [TESS(NC)] Meteorology and Oceanography (METOC) Database. The MAGRID segment provides APIs for the storage and retrieval of grid field data. This software is designed to run under the Defense Information Infrastructure (DII) Common Operating Environment (COE), release 3.1, on a Hewlett-Packard computer running HP-UX 10.20 or a personal computer running the Microsoft Windows NT 4.0 operating system with Service Pack 3.

The software described in this document forms a portion of the METOC Database component of the TESS(NC) Program (Navy Integrated Tactical Environmental Subsystem (NITES) Version I). On 29 October 1996, the Oceanographer of the Navy issued a TESS Program Policy statement in letter 3140 Serial 961/6U570953, modifying the Program by calling for five seamless software versions that are DII COE compliant, preferably to level 5.

The five versions are:

- NITES Version I The local data fusion center and principal METOC analysis and forecast system (TESS(NC))
- NITES Version II The subsystem on the Joint Maritime Command Information System (JMCIS) or Global Command and Control System (GCCS) (NITES/Joint METOC Segment (JMS))
- NITES Version III The unclassified aviation forecast, briefing, and display subsystem tailored to Naval METOC shore activities (currently satisfied by the Meteorological Integrated Data Display System (MIDDS))
- NITES Version IV The Portable subsystem composed of independent Personal Computers (PCs)/workstations and modules for forecaster, satellite, communications, and Integrated Command, Control, Communications, Computer, and Intelligence Surveillance Reconnaissance (IC4ISR) functions (currently the Interim Mobile Oceanographic Support System (IMOSS))
- NITES Version V Foreign Military Sales (currently satisfied by the Allied Environmental Support System (AESS))

NITES I acquires and assimilates various METOC data for use by US Navy and Marine Corps weather forecasters and tactical planners. NITES I provides these users with METOC data, products, and applications necessary to support the warfighter in tactical operations and decision making. NITES I provides METOC data and products to NITES I and II applications, as well as non-TESS(NC) systems requiring METOC data, in a heterogeneous, networked computing environment.

The TESS(NC) Concept of Operations and system architecture require that the METOC Database be distributed both in terms of application access to METOC data and products and in terms of physical location of the data repositories. The organizational structure of the database is influenced by these requirements, and the components of this distributed database are described below.

In accordance with DII COE database concepts, the METOC Database is composed of six DII COE-compliant *shared database* segments. Associated with each shared database segment is an API segment. The segments are arranged by data type as follows:

<u>Data Type</u>	<u>Data Segment</u>	<u>API Segment</u>
Grid Fields	MDGRID	MAGRID
Latitude-Longitude-Time (LLT) Observations	MDLLT	MALLT
Textual Observations and Bulletins	MDTXT	MATXT
Remotely Sensed Data	MDREM	MAREM
Imagery	MDIMG	MAIMG
Climatology Data	MDCLIM	MACLIM

A typical client-server installation is depicted in Figure 1-1 on the next page. This shows the shared database segments residing on a DII COE database server, with a NITES I or II client machine hosting the API segments. Communication between API segments and shared database segments is accomplished over the network using ANSI-standard Structured Query Language (SQL).

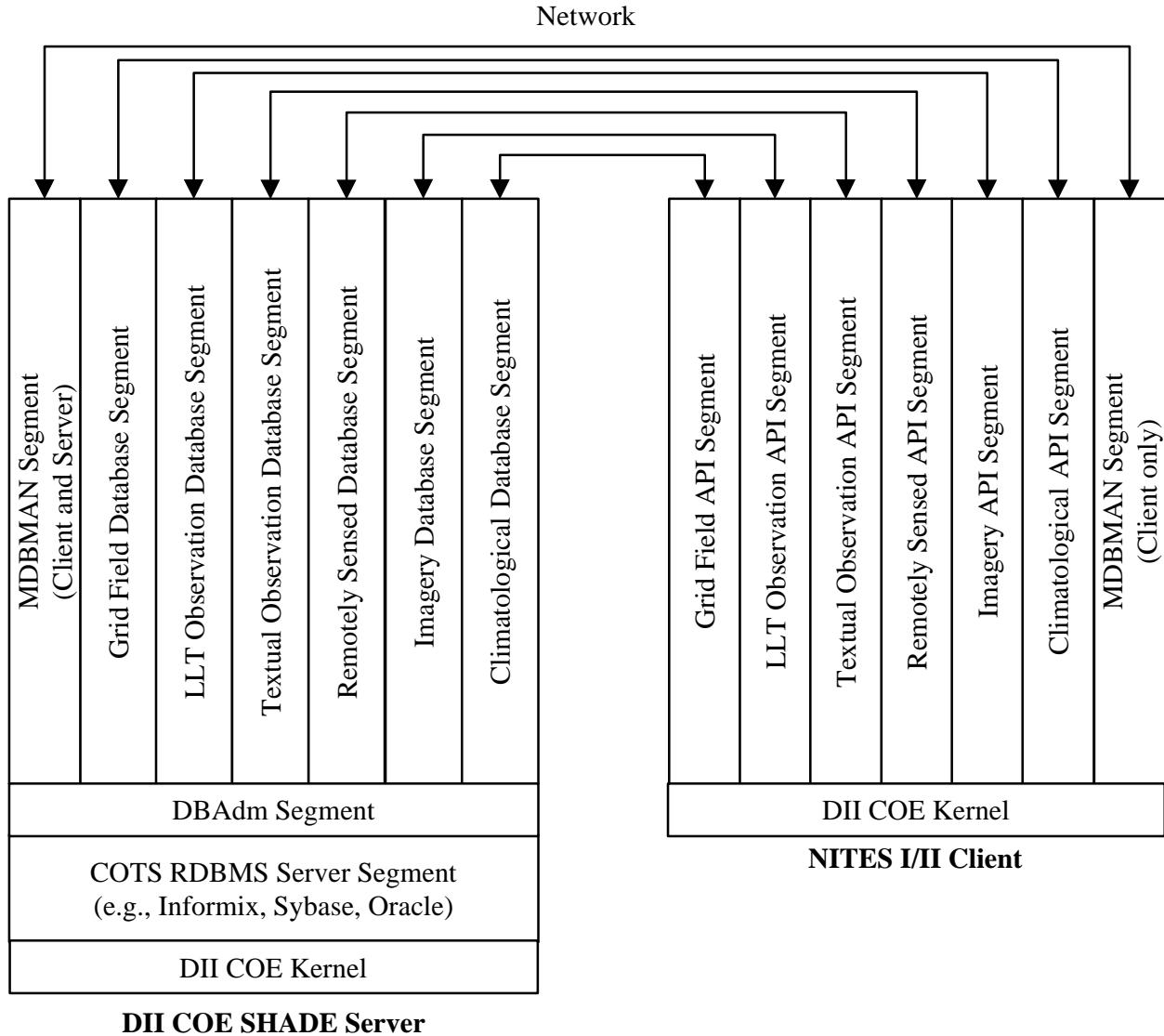


Figure 1-1. TESS(NC) METOC Database Conceptual Organization

The Grid Field Database (MDGRID) segment deals with gridded METOC datasets. These fields provide forecasters with validation information for various atmospheric and oceanographic parameters. A dataset represents a logical collection of discrete grid field data records. The grid data records are logically organized with each other by center, subcenter, and grid model type. A grid data record contains descriptive information (element, level, forecast period, etc.) and the actual grid values.

(This page intentionally left blank.)

2 REFERENCED DOCUMENTS

2.1 Government Documents

STANDARDS

MIL-STD-498
5 December 1994

Software Development and Documentation

SPECIFICATIONS

Unnumbered
30 September 1997

Software Requirements Specification for the Tactical Environmental Support System/Next Century [TESS(3)/NC] Meteorological and Oceanographic (METOC) Database, Space and Naval Warfare Systems Command, Environmental Systems Program Office (SPAWAR PMW-185), Washington, DC

Unnumbered
5 December 1997

Performance Specification (PS) for the Tactical Environmental Support System/Next Century TESS(NC) (AN/UMK-3)

OTHER DOCUMENTS

Unnumbered
02 January 1996

GRIB (Edition 1)
The WMO Format for the Storage of Weather Product
Information and the Exchange of Weather Product Messages
in Gridded Binary Form

U.S. Department of Commerce

National Oceanic and Atmospheric Administration

National Weather Service

National Centers for Environmental Prediction

Clifford H. Dey

NCEP Central C

DII.COE.DocReqs-5
29 April 1997

Defense Information Infrastructure (DII) Common Operating Environment (COE) Developer Documentation Requirements, Version 1.0

DII.COE31.HP10.20.CIP *DII COE V3.1 HP 10.20 Consolidated Installation
Procedures*

DII.3010.HP1020.KernelP2.IG-1 *DII COE Kernel 3.0.1.0P2 Patch 2 for HP-UX 10.20*
30 July 1997 *Installation Guide*

DII.3010.HP1020.KernelP3.IG-1 *DII COE Kernel 3.0.1.0P3 Patch 3 for HP-UX 10.20*
08 August 1997 *Installation Guide*

DII.3010.HP1020.KernelP4.IG-1 *DII COE Kernel 3.0.1.0P4 Patch 4 for HP-UX 10.20*
27 August 1997 *Installation Guide*

Unnumbered
30 September 1997

Database Design Description for the Tactical Environmental Support System/Next Century [TESS(3)/NC] Meteorological and Oceanographic (METOC) Database, Space and Naval Warfare Systems Command, Environmental Systems Program Office (SPAWAR PMW-185), Washington, DC

ipd4600magridrmTES-10
29 January 1999

*Application Program Interface Reference Manual (APIRM)
for the Grid Field API (MAGRID) Segment of the Tactical
Environmental Support System Next Century [TESS(NC)]
Meteorology and Oceanography (METOC) Database*

2.2 Non-Government Documents

World Meteorological Organization, Geneva, Switzerland

WMO-306 *Manual On Codes*
1995

3 SEGMENT OVERVIEW

The MAGRID segment provides APIs used to access Grid Field data in the TESS(NC) METOC Database. The schema for storing these grids is defined by the shared database segment MDGRID. Both of these segments require the Informix Relational Database Management System (RDBMS), version 7.22 (for HP-UX machines) or 7.23 (for Windows NT machines). Both segments run in the DII COE release 3.1, hosted on the following machines and operating systems:

- Tactical Advanced Computer, TAC-3 (HP 750/755)/TAC-4 (HP J201), Operating System: HP-UX 10.20
- IBM-Compatible PC, Operating System: Microsoft Windows NT 4.0, with Service Pack 3

MAGRID uses the following environment variables related to the Informix installation:

- **INFORMIXSERVER** Identifies the Informix server, typically set to `online_coe`
- **INFORMIXDIR** Path to the Informix software, typically `/opt/informix` on HP systems and `C:\informix` on Windows NT systems

The path specified in the `INFORMIXDIR` variable should also be included in the system's PATH variable.

Environmental Variable settings to find MAGRID shared libraries are:

- NT Operating System
`setenv PATH=$PATH;c:/h/MAGRID/bin`
- UNIX Operating System
`setenv SHLIB_PATH /h/MAGRID/bin`

The MACRO definition to correctly set the NT declspec can be defined in one of three ways:

- Makefile using
`DEFINE = -D_MDBDLL`
- In code
`#ifdef _WIN32
#define -D_MDBDLL
#endif`
- Under project settings in the MS Visual Studio

The MAGRID segment is delivered as both archive and runtime libraries on both platforms, with filenames as follows:

<u>Library Type</u>	<u>HP-UX Filename</u>	<u>Windows NT Filename</u>
Archives	libMAGRIDAPI.a	MAGRIDAPI.lib
	libMAGRIDKernel.a	MAGRIDKernel.lib
	libMAGRIDUtils.a	MAGRIDUtils.lib
Runtime	libMAGRIDAPI.sl	MAGRIDAPI.dll
	libMAGRIDKernel.sl	MAGRIDKernel.dll
	libMAGRIDUtils.sl	MAGRIDUtils.dll

Note: Library names have been modified since the programmer's release. Makefiles will need to be updated to reflect the new library.

The runtime libraries are typically installed in the directory /h/MAGRID/bin on HP systems and C:\h\MAGRID\bin on Windows NT systems.

The archive libraries are typically installed in the directory /h/MAGRID/lib on HP systems and C:\h\MAGRID\lib on Windows NT systems.

The individual API methods are described in the APIRM referenced in Section 2. Section 4 of this document provides instructions and programming examples for the use of the APIs.

4 SEGMENT DEVELOPMENT

Programming applications to access grid field data are straightforward. The MAGRID segment provides interfaces to:

- Establish connection to the TESS(NC) MDGRID Database (MAGRIDConnect, MAGRIDRemoteConnect)
- Set the current connection (MAGRIDSetConnection)
- Ingest a 2D Grid Field into the database (MAGRID2DIngest)
- Ingest a 3D Grid Field into the database (MAGRID3DIngest)
- Determine if a model's registration exists in the database (MAGRIDVerifyModel)
- Add a model's registration to the database (MAGRIDRegisterModel)
- Retrieve a registration(s) from the database (MAGRIDRetrRegistration)
- Retrieve a catalog listing of 2D Grid Fields meeting specified criteria from the database (MAGRID2DCatalog)
- Retrieve a catalog listing of 3D Grid Fields meeting specified criteria from the database (MAGRID3DCatalog)
- Retrieve selected Grid Field data from the database (MAGRIDGet2DByQuery)
- Retrieve a single Grid Field from the database (MAGRIDGet2DByID)
- Retrieve a single 3D Grid Field from the database (MAGRIDGetVolumeByID)
- Retrieve a single 3D Grid Field profile (stick) from the database (MAGRIDGetProfileByID)
- Retrieve a single horizontal 3D Grid Field slice from the database (MAGRIDGetSliceByID)
- Retrieve a track of 3D Grid Profiles (sticks) from the database (MAGRIDGetTrack)
- Retrieve a single point from a given grid (MAGRIDGetPoint)
- Delete a selected 2D or 3D Grid Field from the database (MAGRIDDeleteByID)

- Delete one or more 2D or 3D grid fields from a specified dataset based on query criteria supplied
- Update a selected 2D or 3D Grid Field from the database (MAGRIDUpdateByID)
- Free the linked lists returned by MAGRID2DCatalog, MAGRID3DCatalog, and MAGRIDGet2DByQuery (MAGRIDFreeLL)
- Retrieve a center(s) descriptive information from the database (MAGRIDGetCentersInfo)
- Retrieve a model(s) descriptive information from the database (MAGRIDGetModelsInfo)
- Retrieve a parameter(s) descriptive information from the database (MAGRIDGetParametersInfo)
- Retrieve a unit(s) descriptive information from the database (MAGRIDGetUnitsInfo)
- Disconnect from the database (MAGRIDDisconnect, MAGRIDRemoteDisconnect).

Each of these methods is described in detail in the MAGRID APIRM, referenced in Section 2.

4.1 Writing Applications Using the MAGRID APIs

This section shows the use of the MAGRID APIs to perform common data access tasks. In each case, an overview of the actions to be performed is provided, along with a code example and a discussion of pertinent programming concerns.

In all cases, note that the database connect method (MAGRIDConnect or MAGRIDRemoteConnect) must be called to connect the application to the database before any other operations may be performed. The database disconnect method (MAGRIDDisconnect or MAGRIDRemoteDisconnect) should be called to disconnect the application from the database at the end of the session. These methods only need to be called once per session.

All structures must be initialized through use of calloc or memset. Failure to initialize a structure could cause unpredictable results. Garbage in a structure could cause a query to fail.

Section 4.2, MDGRID Database Reference Tables, contains a listing of tables identifying known parameter units, GRIB parameter IDs, models, and geometry. The programmer should reference these tables to verify that proper identifiers are used when formulating grid descriptions.

The MAGRIDRET structure is used to return status information from each MAGRID method. See Section 3.4.18 of the APIRM for details of the information returned in this structure.

4.1.1 Ingesting a 2D Grid Field Into the Database

The MAGRID2DIngest method is used to ingest a 2D Grid Field record into the database. It takes as input a pointer to a MAGRID2DINGEST structure containing the metadata for the grid field and the grid field data to be added. All data in the MAGRID2DINGEST structure should be filled in. **If the lparameterUnit field of the MAGRID2DINGEST structure is set to zero, the default value for the parameter will be used.** See the MDGRID Geophysical Parameter table (Sections 4.2.2, 4.2.3). The code below provides an example of the procedure for adding a grid field to the database.

```
*****
T E S T E R
Purpose: Sample Code for MAGRID2DIngest.
*****
#include <stdio.h>
#include <string.h>
#include <time.h>
#include "MAGRIDAPI.h"
#ifndef _WIN32
#define _MDBDLL
```

```
#endif
void main(argc, argv)
    int argc;
    char **argv;
{
    struct      tm timeptr;
    float       *pData;
    float       rsValue = 0;
    int         x = 0;
    int         y = 0;
    int         j = 0;
    int         XSize, YSize;
    int         nStatus = 0;
    MAGRIDRET  magridRet;
    MAGRID2DINGEST GridData;
    magridRet = MAGRIDConnect();
    if (magridRet.nStatus)
    {
        printf( "\ttester: Error calling MAGRIDConnect (%d) (%s)\n",
                magridRet.nStatus, magridRet.szSQLState );
        printf( "ErrorMessage (%s)\n", magridRet.szErrorMessage );
    }
    else
    {
        printf( "\tMAGRIDConnect : Successful!\n\n");
        memset(&GridData, 0, sizeof(MAGRID2DINGEST));
        memset(&timeptr, 0, sizeof(struct tm));
        GridData.lGeneratingProcId   = 73; /* NORAPS */          */
        GridData.lGridId            = 243; /* Indian Ocean */ */
        GridData.lProductionCenterId = 58; /* US Navy */          */
        GridData.lSubCenterId       = 0;  /* FNMOC */           */
        /* Compute basetime (epoch from Jan 1, 1998) */          */
        timeptr.tm_year = 98;
        timeptr.tm_mon = 1;
        timeptr.tm_mday = 1;
        timeptr.tm_hour = 0;
        GridData.lBaseTime       = (long) mktime (&timeptr);
        GridData.lParameterId    = 7; /*Geopotential Height*/
        GridData.lParameterUnit  = 0; /*geopotential meters*/
        GridData.lLevelType      = 100; /*isobaric level */
        GridData.rsLevelLo       = 500.0;
        GridData.rsLevelHi       = 0.0;
        GridData.lTau             = 24*60; /*24 Hr forecast in mins.*/
        GridData.lQualityIndicator = 0;
        GridData.eDataCategory   = MAGRID_BASE;
        strcpy(GridData.szSecurityClass, "UNCLASS");
        strcpy(GridData.szReceiptMethod, "NET");
        strcpy(GridData.szCompression, "NONE");
        strcpy(GridData.szDescription, "Dummy_Data");
        XSize = 149; /* X dimension for grid id specified */
        YSize = 81; /* Y dimension for the grid id specified */
        GridData.ulSize = sizeof( float ) * XSize* YSize;
        pData = (float *) malloc(GridData.ulSize);

        ****
        /* Populate a dummy */
        ****
        rsValue = 0.0;
        for ( y = 0; y < YSize; y++ )
```

```
{  
    for ( x = 0; x < XSize; x++ )  
    {  
        pData[y*XSize + x] = x;  
    }  
}  
GridData.pGridFieldData = pData;  
/************/  
/* Call MAGRIDStore. */  
/************/  
magridRet = MAGRID2DIngest( &GridData ) ;  
if (magridRet.nStatus)  
{  
    printf( "\tMAGRID2DIngest: Error! (%d) (%s)\n",  
           magridRet.nStatus, magridRet.szSQLState );  
    printf( "\t ErrorMessage (%s)\n", magridRet.szErrorMessage );  
}  
else  
{  
    /* Dataset and record ID for successfully ingested grid */  
    printf( "\tMAGRIDStore: %s Successful (%d)!\n",  
           GridData.szDataSetName,  
           GridData.lRecordId );  
}  
/************/  
/* Call MAGRID_stop. */  
/************/  
magridRet = MAGRIDDisconnect();  
}  
exit( magridRet.nStatus );  
} /* End of main. */
```

4.1.2 Retrieving Center, Model, Parameter, and Unit Information from the Database

The example code following illustrates the use of the four APIs: MAGRIDGetCentersInfo, MAGRIDGetModelsInfo, MAGRIDGetParametersInfo, and MAGRIDGetUnitsInfo.

The MAGRIDGetCentersInfo method will return a linked list of one or more center information records depending on how the lCenterId and lSubcenterId fields are set. Wildcarding (by using MAGRID_QUERY_WILDCARD) any of these two will likely return multiple data records.

The MAGRIDGetModelsInfo method will return a linked list of one or more model information records depending on how the lCenterId, lSubcenterId, lGeneratingProcId, and lGridId fields are set. Wildcarding (by using MAGRID_QUERY_WILDCARD) any of these fields will likely return multiple data records.

The MAGRIDGetParametersInfo method will return a linked list of one or more parameter (elements) information records depending on how the lCenterId, lSubcenterId, and lParameterId fields are set. Wildcarding (by using MAGRID_QUERY_WILDCARD) any of these two will likely return multiple data records. Note that for this method, ParameterIds from 1-127 are

common across all center/subcenter combinations. Anything above 127 is center/subcenter unique. The records returned containing parameter id within the common range will have their lCenterid and lSubcenterId fields set to the constant MAGRID_COMMON_PARAMETER (value 0).

The MAGRIDGetUnitsInfo method will return a linked list of one or more unit information records depending on how the lUnitId is set. Wildcarding (by using MAGRID_QUERY_WILDCARD) this field will return all unit information data records contained in the database.

After calling each routine, an MAGRIDFreeLL method should be used to free the linked list associated with the returned information data records. Also all four routines return a MAGRIDRET return status structure.

```
*****
T   E   S   T   E   R
*****
Purpose: Sample code using MAGRIDCentersInfo, MAGRIDModelsInfo,
          MAGRIDParametersInfo, MAGRIDUnitsInfo APIs
*****  
  

#include <stdlib.h>
#include <stdio.h>
#include "MAGRIDAPI.h"  
  

void main(argc, argv)
    int argc;
    char **argv;
{
    int                      n = 0;
    long                     lNumFound;
    MAGRIDLINKEDLIST         *pInfoLL, InfoLL;
    PMAGRIDUNITDATA          pUnitLL;
    PMAGRIDPARAMETERDATA     pParameterLL;
    PMAGRIDMODELDATA          pModelLL;
    PMAGRIDCENTERDATA         pCenterLL;
    MAGRIDRET                 magridRet;  
  

    magridRet = MAGRIDConnect();  
  

    if (magridRet.nStatus)
    {
        printf( "\tError calling MAGRIDConnect (%d) (%s)\n",
                magridRet.nStatus, magridRet.szSQLState );
        printf( "\tErrorMessage (%s)\n", magridRet.szErrorMessage );
    }
    else
    {
        printf( "\tMAGRIDConnect : Successful!\n" );
```

```
*****  
/* Retrieve All Centers */  
/* known to the database */  
*****  
magridRet = MAGRIDGetCentersInfo  
    ( MAGRID_QUERY_WILDCARD, /* lCenterId */  
      MAGRID_QUERY_WILDCARD, /* lSubCenterId */  
      &lNumFound, &InfoLL );  
  
if (magridRet.nStatus)  
{  
    printf( "Error in MAGRIDGetCentersInfo (%d) (%s)\n",  
           magridRet.nStatus, magridRet.szSQLState );  
}  
else  
{  
    printf( "\tMAGRIDGetCentersInfo : Successful!\n");  
  
*****  
/* Display output values. */  
*****  
if (lNumFound < 1)  
{  
    printf( "\nNo Center entries were found.\n");  
}  
else  
{  
    n = 1;  
    pInfoLL = &InfoLL;  
  
    printf( "\n\nCenter entries found:\n" );  
    while (n <= lNumFound)  
    {  
        pCenterLL = ( PMAGRIDCENTERDATA )pInfoLL->data;  
  
        printf( "\n\tCenter ID: %d\n",  
               pCenterLL->lProductionCenterId);  
        printf( "\tSub Center ID: %d\n", pCenterLL->lSubCenterId);  
        printf( "\tCenter Name: %s\n", pCenterLL->szCenterName);  
        n++;  
  
        pInfoLL = pInfoLL->next;  
    }  
    MAGRIDFreeLL( &InfoLL );  
}  
}  
  
*****  
/* Retrieve the Unit Record */  
/* for Unit Id 200 */  
*****  
magridRet = MAGRIDGetUnitsInfo ( 200, /* lUnitId */  
                               &lNumFound, &InfoLL );  
if (magridRet.nStatus)  
{  
    printf( "Error in MAGRIDGetUnitsInfo (%d) (%s)\n",  
           magridRet.nStatus, magridRet.szSQLState );  
}  
else  
{
```

```
*****  
/* Display output values. */  
*****  
if (lNumFound < 1)  
{  
    printf("\nNo unit entries were found.\n");  
}  
else  
{  
    printf("\n\nUnit entries found:\n");  
    n = 1;  
    pInfoLL = &InfoLL;  
  
    while (n <= lNumFound)  
    {  
        pUnitLL = ( PMAGRIDUNITDATA )pInfoLL->data;  
  
        printf("\n\tUnit ID: %d\n",pUnitLL->lUnitId);  
        printf("\tUnit Name: %s\n",pUnitLL->szUnitName);  
        printf("\tUnit Abbrev Name: %s\n",pUnitLL->szUnitAbrv);  
        n++;  
  
        pInfoLL = pInfoLL->next;  
    }  
    MAGRIDFreeLL( &InfoLL );  
}  
  
*****  
/* Retrieve the List of Models */  
/* produced by FNMOc (center id 58 */  
/* subcenter id 0) */  
*****  
magridRet = MAGRIDGetModelsInfo ( 58, 0,  
                                  MAGRID_QUERY_WILDCARD, /* lModelId */  
                                  MAGRID_QUERY_WILDCARD, /* lGridId */  
                                  &lNumFound, &InfoLL );  
  
if (magridRet.nStatus)  
{  
    printf( "Error in MAGRIDGetModelsInfo (%d) (%s)\n",  
           magridRet.nStatus, magridRet.szSQLState );  
}  
else  
{  
    *****  
    /* Display output values. */  
    *****  
    if (lNumFound < 1)  
    {  
        printf("\nNo Model entries were found.\n");  
    }  
    else  
{  
  
        printf("\n\nModel entries found:\n");  
        n = 1;  
        pInfoLL = &InfoLL;  
  
        while (n <= lNumFound)  
        {
```

```
pModelLL = ( PMAGRIDMODELDATA )pInfoLL->data;
printf("\n\tlProductionCenterId = %d \n",
      pModelLL->lProductionCenterId);
printf("\tlSubCenterId = %d \n",
      pModelLL->lSubCenterId);
printf("\tlGeneratingProcId = %d \n",
      pModelLL->lGeneratingProcId);
printf("\tlGridId = %d \n",
      pModelLL->lGridId);
printf("\tszCenterName = %s \n",
      pModelLL->szCenterName);
printf("\tszmodelName = %s \n",
      pModelLL->szmodelName);

n++;

pInfoLL = pInfoLL->next;
}
MAGRIDFreeLL( &InfoLL );
}

/*****
/* Retrieve the List of Parameters */
/* produced by FNMOc (center id 58 */
/* subcenter id 0) */
*****
magridRet = MAGRIDGetParametersInfo ( 58, 0,
                                      MAGRID_QUERY_WILDCARD, /* lParameterId */
                                      &lNumFound, &InfoLL );
if (magridRet.nStatus)
{
    printf( "Error in MAGRIDGetParametersInfo (%d) (%s)\n",
            magridRet.nStatus, magridRet.szSQLState );
}
else
{
    printf( "\tMAGRIDtGetParametersInfo : Successful!\n" );

/*****
/* Display output values. */
*****
if (lNumFound < 1)
{
    printf("\nNo Parameter entries were found.\n");
}
else
{
    printf("\n\nParameter entries found:\n");
    n = 1;
    pInfoLL = &InfoLL;

    while (n <= lNumFound)
    {
        pParameterLL = ( PMAGRIDPARAMETERDATA )pInfoLL->data;
        printf("\n\tlProductionCenterId = %d \n",
              pParameterLL->lProductionCenterId);
        printf("\tlSubCenterId = %d \n",
              pParameterLL->lSubCenterId);
        printf("\tlParameterId = %d \n",
              pParameterLL->lParameterId);
    }
}
```

```
    pParameterLL->lParameterId);
    printf("\tszParameterName = %s \n",
           pParameterLL->szParameterName);
    printf("\trsMinValidValue = %f \n",
           pParameterLL->rsMinValidValue);
    printf("\trsMaxValidValue = %f \n",
           pParameterLL->rsMaxValidValue);
    printf("\tlDefaultUnitId = %d \n",
           pParameterLL->lDefaultUnitId);
    printf("\tszDefaultUnitAbrv = %s \n",
           pParameterLL->szDefaultUnitAbrv);
    printf("\tszDefaultUnitName = %s \n",
           pParameterLL->szDefaultUnitName);

    n++;

    pInfoLL = pInfoLL->next;
}
MAGRIDFreeLL( &InfoLL );
}

/*
/* Call MAGRIDDisconnect. */
*/
magridRet = MAGRIDDisconnect();
}

printf( "Exiting (%d).\n", magridRet.nStatus );
exit( magridRet.nStatus );
}
```

4.1.3 Ingesting a 3D Grid Field Into the Database

The MAGRID3DIgest method is used to ingest a 3D Grid Field record into the database. It takes as input a pointer to a MAGRID3DINGEST structure containing the metadata for the grid field and the grid field data to be added. All data in the MAGRID3DINGEST structure should be filled in. **If the lparameterUnit field of the MAGRID3DINGEST structure is set to zero, the default value for the parameter will be used.** See the MDGRID Geophysical Parameter tables (Sections 4.2.2, 4.2.3). The code below provides an example of the procedure for registering (dynamic registration with an unknown GRIB Grid Id) and adding a 3D grid field to the database.

```
*****
T   E   S   T   E   R
*****
Purpose: Sample Code using MAGRID3DIgest.
*****
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <time.h>
#include "MAGRIDAPI.h"

#define MINUTES_PER_HOUR 60
#ifndef _WIN32
```

```
#define _MDBDLL
#ifndef _MDBDLL
#endif

/*
***** M A I N *****
*/
void main(argc, argv)
    int argc;
    char **argv;
{

    struct tm    timeptr;
    float        *pData;
    float        rsValue = 0;
    int          x = 0;
    int          y = 0;
    int          z = 0;
    int          index = 0;
    int          XSize, YSize;
    int          nStatus = 0;
    int          lProductionCenterId, lSubCenterId, lGeneratingProcId;
MAGRIDRET    magridRet;
MGRID3DINGEST GridData;
MGRIDINPUTREG RegData;
PMAGRIDSPHERICAL pSphericalData;

magridRet = MAGRIDConnect();
if (magridRet.nStatus)
{
    printf(
        "\ttester:Error calling MAGRIDConnect Status (%d) SQLState(%s)\n",
        magridRet.nStatus, magridRet.szSQLState );
    printf( "\tErrorMessage (%s)\n", magridRet.szErrorMessage );
}
else
{
    printf( "\tMAGRIDConnect : Successful!\n\n" );

/*
***** Register the MODAS 3-D Grid with the database. *****
*/
lProductionCenterId = 58;
lSubCenterId       = 0;
lGeneratingProcId  = 200;

/* Fill in Registration information for 3D example */
memset(&RegData, 0, sizeof(MGRIDINPUTREG));
RegData.lGridId = MAGRID_UNDEFINED_GRIDID;
RegData.lProductionCenterId = lProductionCenterId;
RegData.lSubCenterId = lSubCenterId;
RegData.lGeneratingProcId = lGeneratingProcId;

RegData.rsRegLat = 35.0;
RegData.rsRegLon = -120.0;
RegData.rsRegX = 1.0;
RegData.rsRegY = 1.0;

RegData.rsXDistance = 224.0;
```

```
RegData.rsYDistance = 224.0;

RegData.lMaxXPoint = 96;
RegData.lMaxYPoint = 100;
RegData.lMaxZPoint = 34;
RegData.eScanMode = MAGRID_pXinpY;

RegData.stProjectionDesc.eProjection = MAGRID_SPHERICAL;

pSphericalData = (PMAGRIDSPHERICAL) &RegData.stProjectionDesc.projParms;
pSphericalData->rsXResolution = 1.0;
pSphericalData->rsYResolution = 1.0;

/*****************/
/* Call MAGRIDRegisterModel. */
/*****************/
magridRet = MAGRIDRegisterModel( &RegData ) ;

if (magridRet.nStatus)
{
    printf( "\tMAGRIDRegisterModel: Error (%d)!!\n",
            magridRet.nStatus );
}
else
{
    printf("\tMAGRIDRegisterModel: Successful!!\n");
    printf("\t\tGridID of registration is %d\n", RegData.lGridId);
}

if (!magridRet.nStatus)
{
    memset(&GridData, 0, sizeof(MAGRID3DINGEST));
    GridData.lGeneratingProcId = lGeneratingProcId;
    GridData.lGridId           = RegData.lGridId;      /* New Dynamic
                                                       Grid ID */
    GridData.lProductionCenterId = lProductionCenterId;
    GridData.lSubCenterId = lSubCenterId;

    /* Compute basetime (epoch) from Jan 1, 1998 */
    memset(&timeptr, 0, sizeof(struct tm));
    timeptr.tm_year = 98;
    timeptr.tm_mon = 1;
    timeptr.tm_mday = 1;
    timeptr.tm_hour = 0;

    GridData.lBaseTime        = (long) mktime(&timeptr); /*MODAS Sound
                                                       Velocity*/
    GridData.lParameterId     = 251;
    GridData.lParameterUnit   = 0;
    GridData.lLevelType       = 160;
    GridData.lTau              = 24 * MINUTES_PER_HOUR; /* 24 hour
                                                       forecast */

    GridData.lQualityIndicator = 0;
    GridData.eDataCategory     = MAGRID_BASE;
    strcpy(GridData.szSecurityClass, "UNCLASS");
    strcpy(GridData.szReceiptMethod, "NET");
    strcpy(GridData.szCompression, "NONE");
    strcpy(GridData.szDescription, "Dummy_Data");

    XSize = RegData.lMaxXPoint;                         /* X dimension of grid */
}
```

```
YSize = RegData.lMaxYPoint; /* Y dimension of grid */
GridData.lMaxZPoint = RegData.lMaxZPoint; /* Number of levels */

GridData.ulSize = sizeof( float ) * XSize * YSize * GridData.lMaxZPoint;

pData = (float *) malloc(GridData.ulSize);

/*****************/
/* Populate a dummy grid and fill in level data */
/*****************/
rsValue = 0.0;
index = 0;

for ( z = 0; z < GridData.lMaxZPoint; z++ )
{
    GridData.rsLevel[z] = z * 100.0;

    for ( y = 0; y < YSize; y++ )
    {
        for ( x = 0; x < XSize; x++ )
        {
            pData[index] = (float) (z*XSize*YSize) + (y*XSize) + x;
        }
    }
}

GridData.pGridFieldData = pData;

/*****************/
/* Call MAGRID3DIngest. */
/*****************/
magridRet = MAGRID3DIngest( &GridData ) ;

if (magridRet.nStatus)
{
    printf( "\tMAGRID3DIngest: Error! Status (%d) SQLState (%s)\n",
           magridRet.nStatus, magridRet.szSQLState );
    printf( "\tErrorMessage (%s)\n", magridRet.szErrorMessage );

}
else
{
    printf( "\tMAGRID3DIngest: Successful (%s:%d)!\n",
           GridData.szDataSetName, GridData.lRecordId);

}
free(pData);
}

/*****************/
/* Call MAGRID_stop. */
/*****************/
magridRet = MAGRIDDisconnect( );

}

exit( magridRet.nStatus );
} /* End of main. */
```

4.1.4 Registering Grid Models Into the Database

The registration routines, MAGRIDVerifyModel and MAGRIDRegisterModel, support both 2D and 3D grid registrations. The MAGRIDVerifyModel method is used to determine if the database has support for a given model being distributed from a production center. The MAGRIDVerifyModel method takes as input the GRIB Production Center ID, the GRIB Production Sub-center ID, the GRIB Grid ID, and the GRIB Generating Process ID. The MAGRIDRegisterModel method may be used to register the model with the database, if not currently supported. It takes as input a pointer to a MAGRIDINPUTREG structure containing the geometric descriptions of the grid size, spacing, area of coverage, projection type, and grid scan mode. The code below provides an example of the procedure for determining if a 2D model is supported and then updating the database in order to provide the desired support. A second example within this code shows how to register dynamic grids (grids that are dynamically created that do not have a registered GRIB identifier).

```
*****
          T   E   S   T   E   R
Purpose: Sample Code for MAGRIDRegister.
*****  
*****  
#include <stdio.h>
#include <string.h>
#include "MAGRIDAPI.h"  
  
void main(argc, argv)
    int argc;
    char **argv;
{
    long lProductionCenterId = 58; /* US. Navy */
    long lSubCenterId       = 0;
    long lGridId            = 23;
    long lGeneratingProcId = 28;
    MAGRIDRET magridRet;
    MAGRIDINPUTREG RegData;
    PMAGRIDSFERICAL pSphericalData;
    magridRet = MAGRIDConnect();
    if (magridRet.nStatus)
    {
        printf( "\ttester: Error calling MAGRIDConnect (%d) (%s)\n",
                magridRet.nStatus, magridRet.szSQLState );
        printf( "ErrorMessage (%s)\n", magridRet.szErrorMessage );
    }
    else
    {
        ****
        /* Grid 2D example with a center */
        /* defined Grid ID */
        ****
        printf( "\tMAGRIDConnect : Successful!\n\n" );
    }
}
```

```
memset(&RegData, 0, sizeof(MAGRIDINPUTREG));
/*****************/
/* Call MAGRIDVerifyModel */
/* To see if DB knows about model */
/*****************/
magridRet = MAGRIDVerifyModel( lProductionCenterId, lSubCenterId,
                                lGridId, lGeneratingProcId);

if (magridRet.nStatus == UNKNOWN_REGISTRATION)
{
    RegData.lGridId = lGridId;
    RegData.lProductionCenterId = lProductionCenterId;
    RegData.lSubCenterId = lSubCenterId;
    RegData.lGeneratingProcId = lGeneratingProcId;
    RegData.rsRegLat = 60.0;
    RegData.rsRegLon = 90.0;
    RegData.rsRegX = 1.0;
    RegData.rsRegY = 1.0;
    RegData.rsXDistance = 224.0;
    RegData.rsYDistance = 224.0;
    RegData.lMaxXPoint = 96;
    RegData.lMaxYPoint = 100;
    strcpy (RegData.szRegName, "Gobal");
    strcpy (RegData.szMadelName, "TOPS");
    strcpy (RegData.CenterName, "Navy-FNOC");
    RegData.eScanMode = MAGRID_pXinpY;
    RegData.stProjectionDesc.eProjection = MAGRID_SPHERICAL;
    pSphericalData = (PMAGRIDSUPERICAL) &RegData.stProjectionDesc.projParms;
    pSphericalData->rsXResolution = 1.0;
    pSphericalData->rsYResolution = 1.0;
    /* Call MAGRIDRegisterModel. */
    /* */
magridRet = MAGRIDRegisterModel( &RegData ) ;
if (magridRet.nStatus)
{
    printf( "\tMAGRIDRegisterModel: Error (%d)!!\n",
            magridRet.nStatus );
    printf( "ErrorMessage (%s)\n", magridRet.szErrorMessage);

}
else if (magridRet.nStatus)
{
    printf( "\tMAGRIDVerifyModel: Error (%d)!!\n", magridRet.nStatus );
    printf( "ErrorMessage (%s)\n", magridRet.szErrorMessage);

}
else
{
    printf( "\t\tRegistration NOT needed\n");
}

/* Grid 3D example without a GRIB */
/* defined Grid ID(user-defined) */
/*****************/
lProductionCenterId = 58;
lSubCenterId        = 0;
```

```
lGridId          = MAGRID_UNDEFINED_GRIDID; /* dynamic registration */
lGeneratingProcId = 24;

/* Fill in Registration information for 3D example */
RegData.lGridId = lGridId;
RegData.lProductionCenterId = lProductionCenterId;
RegData.lSubCenterId = lSubCenterId;
RegData.lGeneratingProcId = lGeneratingProcId;

RegData.rsRegLat = 35.0;
RegData.rsRegLon = -120.0;
RegData.rsRegX = 1.0;
RegData.rsRegY = 1.0;

RegData.rsXDistance = 224.0;
RegData.rsYDistance = 224.0;
RegData.lMaxXPoint = 96;
RegData.lMaxYPoint = 100;
RegData.lMaxZPoint = 34;
RegData.eScanMode = MAGRID_pXinpY;

RegData.stProjectionDesc.eProjection = MAGRID_SPHERICAL;

pSphericalData = (PMAGRIDSPHERICAL)
                  &RegData.stProjectionDesc.projParms;
pSphericalData->rsXResolution = 1.0;
pSphericalData->rsYResolution = 1.0;

/*****************/
/* Call MAGRIDRegisterModel. */
/*****************/
magridRet = MAGRIDRegisterModel( &RegData ) ;

if (magridRet.nStatus)
{
    printf( "\tMAGRIDRegisterModel: Error (%d)!!\n",
            magridRet.nStatus );
}
else
{
    printf("\tMAGRIDRegisterModel: Successful!!\n");
    printf("\t\tGridID of registration is %d\n", RegData.lGridId);
}

/*****************/
/* Call MAGRIDDisconnect. */
/*****************/
magridRet = MAGRIDDisconnect();
}
exit( magridRet.nStatus );
} /* End of main. */
```

4.1.5 Deleting Single Grid Fields From the Database

The MAGRIDDeleteByID method is used to delete 2D and 3D grid fields from the database. This method requires the dataset name and the record id for the grid field to be deleted. Typically the MAGRID2DCatalog or MAGRID3DCatalog method is used to get a list of the

candidate records for deletion, as shown in the example below. Once the list has been retrieved, the program can cycle through the list and call MAGRIDDeleteByID to delete the current element.

```
*****  
T   E   S   T   E   R  
Purpose: Sample code for MAGRIDDeleteByID.  
*****  
#include <stdlib.h>  
#include <stdio.h>  
#include "MAGRIDAPI.h"  
#define SECONDS_PER_DAY    24*60*60  
#define MINUTES_PER_DAY    24*60  
#ifdef _WIN32  
#define _MDBDLL  
#endif  
void main(argc, argv)  
    int argc;  
    char **argv;  
{  
    MAGRIDLINKEDLIST    GridFieldDescLL;  
    PMAGRIDLINKEDLIST  pGridFieldDescLL;  
    PMAGRIDFIELDDESC   pLL;  
    int                 n = 0;  
    long                lNumFound;  
    MAGRIDRET          magridRet;  
    MAGRIDQUERY        GridQuery;  
    magridRet = MAGRIDConnect();  
    if (magridRet.nStatus)  
    {  
        printf( "\tError calling MAGRIDConnect (%d) (%s)\n",  
                magridRet.nStatus, magridRet.szSQLState );  
        printf( "\tErrorMessage (%s)\n", magridRet.szErrorMessage );  
    }  
    else  
    {  
        printf( "\tMAGRIDConnect : Successful!\n");  
        memset(&GridQuery, 0, sizeof(MAGRIDQUERY));  
        GridQuery.lProductionCenterId = 58; /* US Navy */  
        GridQuery.lSubCenterId       = 0; /* FNMOC */  
        GridQuery.lGeneratingProcId = MAGRID_QUERY_WILDCARD;  
        GridQuery.lGridId           = MAGRID_QUERY_WILDCARD;  
        GridQuery.lParameterId      = MAGRID_QUERY_WILDCARD;  
                                /*Two Days*/  
        GridQuery.lBeginBaseTime   = time (0) - 2*SECONDS_PER_DAY;  
        GridQuery.lEndBaseTime    = MAGRID_QUERY_WILDCARD;  
        GridQuery.lBeginTau        = 0; /* TAU in minutes */  
        GridQuery.lEndTau          = MINUTES_PER_DAY; /*TAU in minutes (24hr)*/  
        GridQuery.rsBeginLevel    = MAGRID_QUERY_WILDCARD;  
        GridQuery.rsEndLevel      = 0.0; /* Used if Layer Type */  
        GridQuery.lLevelType      = 100; /* isobaric level */  
        GridQuery.stGeoArea.rsNLat = 90.0;  
        GridQuery.stGeoArea.rsSLat = -90.00;  
        GridQuery.stGeoArea.rsWLon = -180.00;  
        GridQuery.stGeoArea.rsELon = 180.00;  
    }  
}
```

```
*****  
/* Retrieve Grid Catalog.*/  
*****  
magridRet = MAGRID2DCatalog ( &GridQuery, &lNumFound, &GridFieldDescLL );  
if (magridRet.nStatus)  
{  
    DPRINT( "tester: Error in MAGRIDRetrCat (%d) (%s)\n",  
           magridRet.nStatus, magridRet.szSQLState );  
}  
else  
{  
  
    *****  
    /* Display output values. */  
    *****  
    if (lNumFound < 1)  
    {  
        printf( "\nNo catalog entries were found.\n" );  
    }  
    else  
    {  
        n = 1;  
        pGridFieldDescLL = &GridFieldDescLL;  
        while (n <= lNumFound)  
        {  
            PLL = ( PMAGRIDFIELDDESC )pGridFieldDescLL->data;  
            printf("Model ID: %d\n",PLL->lGeneratingProcId);  
            printf("Registration ID: %d\n",PLL->lGridId);  
            printf("Production Center ID: %d\n",  
                   PLL->lProductionCenterId);  
            printf("SubCenter ID: %d\n",PLL->lSubCenterId);  
            printf("Parameter ID: %d\n",PLL->lParameterId);  
            printf("Parameter Unit ID: %d\n",  
                   PLL->lParameterUnit);  
            printf("Level Type ID: %d\n",PLL->lLevelType);  
            printf("Base Time: %d\n",PLL->lBaseTime);  
            printf("Dataset Ref Name: %s\n",PLL->szDataSetName);  
            printf("Record ID: %d\n",PLL->lRecordId);  
            printf("Tau: %ld\n",PLL->lTau);  
            printf("Level Lo: %f\n",PLL->rsLevelLo);  
            printf("Level Hi: %f\n",PLL->rsLevelHi);  
            printf("Quality Indicator: %ld\n",  
                   PLL->lQualityIndicator);  
            printf("Data Category: %ld\n",PLL->lDataCategory);  
            printf("Security Class: %s\n",PLL->szSecurityClass);  
            printf("Receipt Method: %s\n",PLL->szReceiptMethod);  
            printf("Compression: %s\n",PLL->szCompression);  
            printf("Projection: %d\n",PLL->eProjection);  
  
        }  
    }  
    *****  
    /* Delete Dataset Record. */  
    *****  
    magridRet = MAGRIDDeleteByID ( PLL->szDataSetName,  
                                     PLL->lRecordId );  
    if (magridRet.nStatus)  
    {  
        printf("Unable to Delete: %s %d (error:%d)\n",  
               PLL->szDataSetName, PLL->lRecordId,  
               magridRet.nStatus);  
    }  
}
```

```
        else
    {
        printf("Successfully Deleted: %s %d\n",
               pLL->szDataSetName, pLL->lRecordId);
    }
    n++;
    pGridFieldDescLL = pGridFieldDescLL->next;
}
}
MAGRIDFreeLL( &GridFieldDescLL );
/*
 * Call MAGRID_Disconnect.*/
*/
magridRet = MAGRIDDisconnect();
}
DPRINT( "tester: Exiting (%d).\n", magridRet.nStatus );
exit( magridRet.nStatus );
}
```

4.1.6 Deleting Multiple Grids From a 2D or 3D Dataset

The MAGRIDDeleteByQuery method is used to delete multiple grids from a single 2D or 3D dataset. A grid dataset is considered to be a group of groups generated at the same center/subcenter using the same model and registration (grid spacing and Area of Interest (AOI)). This method takes as input a MAGRIDDELETEQUERY structure containing criteria for deleting the 2D or 3D grids. The MAGRIDDELETEQUERY structure allows wildcarding of the IParameterId, IBeginBaseTime, IEndBaseTime, IBeginTau, IEndTau, IBeginReceiptTime, and IEndReceiptTime fields. The eGridType, IProductCenterId, ISubCenterId, IGeneratingProcId, and IGridId may not be wildcarded since it is these fields that define a single 2D or 3D dataset. The following example shows a program that deletes all the NOGAPS Global 2.5 degree grids that have a basetime older than 2 days past.

```
*****
T   E   S   T   E   R
Purpose: Sample code using MAGRIDDeleteByQuery.
*****
#include <stdlib.h>
#include <stdio.h>
#include "MAGRIDAPI.h"
#define SECS_PER_DAY 24*60*60
#define MINUTES_PER_DAY 24*60
void main(argc, argv)
    int argc;
    char **argv;
{
    long                 lNumberDeleted;
    MAGRIDRET            magridRet;
    MAGRIDDELETEQUERY    GridQuery;
    magridRet = MAGRIDConnect();
    if (magridRet.nStatus)
    {
        printf("\tError calling MAGRIDConnect (%d) (%s)\n",
               magridRet.nStatus, magridRet.szSQLState);
```

```
    printf("\tErrorMessage (%s)\n", magridRet.szErrorMessage);
}
else
{
    printf("\tMAGRIDConnect : Successful!\n");
    /* Query will delete all NOGAPS Global 2.5 degree grids that have
       a basetime older than 2 days ago */
    memset(&GridQuery, 0, sizeof(MAGRIDDELETEQUERY));
    GridQuery.eGridType          = MAGRID_2D;
    GridQuery.lProductionCenterId = 58;      /* US Navy           */
    GridQuery.lSubCenterId       = 0;        /* FNMOc            */
    GridQuery.lGeneratingProcId = 58;      /* NOGAPS           */
    GridQuery.lGridId           = 223;     /* Global 2.5 Degree */
    GridQuery.lParameterId      = MAGRID_QUERY_WILDCARD;
    GridQuery.lBeginBaseTime    = MAGRID_QUERY_WILDCARD;
    GridQuery.lEndBaseTime      = time(0) - 2*SECS_PER_DAY;
    GridQuery.lBeginTau         = MAGRID_QUERY_WILDCARD;
    GridQuery.lEndTau           = MAGRID_QUERY_WILDCARD;
    GridQuery.lBeginReceiptTime = MAGRID_QUERY_WILDCARD;
    GridQuery.lEndReceiptTime   = MAGRID_QUERY_WILDCARD;

    /*****
    /* Delete the Grids.*/
    *****/
    magridRet = MAGRIDDeleteByQuery (&GridQuery, &lNumberDeleted);
    if (magridRet.nStatus)
    {
        printf("Error in MAGRIDDeleteByQuery (%d) (%s)\n",
               magridRet.nStatus, magridRet.szSQLState);
        printf("Error Message (%s)\n", magridRet.szErrorMessage);
    }
    else
    {
        printf("\tMAGRIDDeleteByQuery : %d Records Successfully Deleted!\n",
               lNumberDeleted);
    }
    /*****
    /* Call MAGRID_Disconnect.*/
    *****/
    magridRet = MAGRIDDisconnect();
}
printf("Exiting (%d).\n", magridRet.nStatus);
exit(magridRet.nStatus);
}
```

4.1.7 Retrieving 2D Grid Fields From the Database

Two methods are provided for retrieving 2D grid field data. The MAGRIDGet2DByQuery method takes as input a MAGRIDQUERY structure containing criteria for 2D grids to be retrieved. It returns a linked list of MAGRIDDATA structures containing the data for 2D grid fields that matched the input criteria, the number of matching grids found, and a MAGRIDRET return status structure. This method retrieves all grids matching the input criteria. Note that if MAGRIDGet2DByQuery is called, MAGRIDFreeLL must be called afterwards to free the linked list returned, and then the head may also need to be freed if dynamically allocated.

The second method for retrieving grid fields is MAGRIDGet2DByID. This method takes as input the datasetname and record id of the grid field record to be retrieved, and returns a single MAGRIDDATA structure containing the retrieved data and a MAGRIDRET return status structure. This method requires knowledge of the datasetname and the record to be retrieved, which is typically found by calling MAGRID2DCatalog to get a list of grid fields matching specific criteria. The advantage of using the MAGRIDGet2DByID is that the program can perform additional operations on the catalog to find a specific grid to retrieve. For example, one might retrieve a catalog listing of grid fields for a specific parameter within a specific time frame, then retrieve the latest forecast from the list. The MAGRIDDATA structure that is returned contains a pointer to the gridfield data. This pointer should be freed after use, and then the MAGRIDDATA structure should be freed if dynamically allocated.

The code example below shows the use of both MAGRIDGet2DByQuery and MAGRIDGet2DByID to retrieve 2D grid fields and also provides a further example of the use of MAGRID2DCatalog.

```
*****
T E S T E R
Purpose: Test driver for MAGRIDGet2DByQuery and
MAGRIDGet2DByID.
*****
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <time.h>
#include "MAGRIDAPI.h"
#define SECONDS_PER_DAY    24*60*60
#define MINUTES_PER_DAY    24*60
#ifndef _WIN32
#define _MDBDLL
#endif

void main(argc, argv)
int argc;
char **argv;
{
    int             n = 0;
    MAGRIDRET      magridRet;
    MAGRIDQUERY    GridQuery;
    MAGRIDREFERENCE GridRefData;
    MAGRIDDATA     stGridData, pGridData;
    PMAGRIDFIELDDESC pGridFieldDesc = NULL;
    MAGRIDFORMAT   stGridFormat;
    long           lNumFound =0;
    MAGRIDLINKEDLIST GridDataLL, *pGridDataLL,
                      GridFieldDescLL, *pGridFieldDescLL;
    magridRet = MAGRIDConnect();
    if (magridRet.nStatus)
    {
        printf( "\tError calling MAGRIDConnect (%d) (%s)\n", magridRet.nStatus,
               magridRet.szSQLState );
        printf( "ErrorMessage (%s)\n", magridRet.szErrorMessage );
```

```
        exit ( 0 );
    }
else
{
    /*****
     * Populate the catalog query structure *
     *****/
    memset(&GridQuery, 0, sizeof(MAGRIDQUERY));
    GridQuery.lGeneratingProcId = 73;           /* NORAPS */          */
    GridQuery.lProductionCenterId = 58;          /* US Navy */         */
    GridQuery.lSubCenterId = 0;                  /* FNMOc */          */
    GridQuery.lGridId = MAGRID_QUERY_WILDCARD;

    GridQuery.lParameterId = MAGRID_QUERY_WILDCARD;
    GridQuery.lBeginBaseTime = MAGRID_QUERY_WILDCARD;
    GridQuery.lEndBaseTime = MAGRID_QUERY_WILDCARD;
    GridQuery.lBeginTau = 0;                      /* TAU in minutes */   /
    GridQuery.lEndTau = 1440;                     /* TAU in minutes (24hr) *//
    GridQuery.rsBeginLevel = MAGRID_QUERY_WILDCARD;
    GridQuery.rsEndLevel = MAGRID_QUERY_WILDCARD;
    GridQuery.lLevelType = MAGRID_QUERY_WILDCARD;
    GridQuery.stGeoArea.rsNLat = 90.0;
    GridQuery.stGeoArea.rsSLat = -90.0;
    GridQuery.stGeoArea.rsWLon = -180.0;
    GridQuery.stGeoArea.rsELon = 180.0;
    GridQuery.eDataCategory = MAGRID_GET_ALL;
    GridQuery.lBeginReceiptTime = MAGRID_QUERY_WILDCARD;
    GridQuery.lEndReceiptTime = MAGRID_QUERY_WILDCARD;
    /*****
     * Populate the grid format structure *
     *****/
    stGridFormat.eOutputFormat = MAGRID_GET_AS_SPECIFIED;
    stGridFormat.lMaxXPoint = 105;
    stGridFormat.lMaxYPoint = 133;
    stGridFormat.eScanMode = MAGRID_pXinnY;
    stGridFormat.eProjection = MAGRID_SPHERICAL;
    stGridFormat.lUnitId = 650;
    /*****
     * Call the API MAGRIDGet2DByQuery to retrieve multiple Grids */
    /*****
magridRet = MAGRIDGet2DByQuery ( &GridQuery, &stGridFormat,
                                &lNumFound, &GridDataLL );
if ( magridRet.nStatus )
{
    printf( "Error in MAGRIDGet2DByQuery (%d) (%s)\n",
            magridRet.nStatus, magridRet.szSQLState );
    printf( "ErrorMessage (%s)\n", magridRet.szErrorMessage);

}
else
{
    printf( "\tMAGRIDGet2DByQuery : Successful!\n");
    /*****
     * Display output values. */
    /*****
if ( lNumFound < 1)
{
    printf( "\nNo grids were found.\n");
}
```

```
    else
    {
        n = 1;
        pGridDataLL = &GridDataLL;
        while (n <= lNumFound)
        {
            pGridData = ( PMAGRIDDATA )
                        pGridDataLL->data;
            printf("\n");
            printf("Dataset Name:%s\n",
                   pGridData.stGridFieldDesc->szDataSetName);
            printf("Record Id: %d\n",
                   pGridData.stGridFieldDesc->lRecordId);
            n++;
            pGridDataLL = pGridDataLL->next;
        }
        MAGRIDFreeLL( &GridDataLL );
    }

/*****************************************/
/* Demonstrate the Second way to retrieve grids */
/* Retrieve Grid Catalog. */
/*****************************************/
magridRet = MAGRID2DCatalog ( &GridQuery, &lNumFound,
                             &GridFieldDescLL );

if (magridRet.nStatus)
{
    printf( "Error in MAGRID2DCatalog (%d) (%s)\n",
            magridRet.nStatus, magridRet.szSQLState );
    printf( "ErrorMessage (%s)\n", magridRet.szErrorMessage);

}
else
{

/*****************************************/
/* Display output values. */
/*****************************************/
if (lNumFound < 1)
{
    printf("\nNo catalog entries were found.\n");
}
else
{
    n = 1;
    pGridFieldDescLL = &GridFieldDescLL;
    while (n <= lNumFound)
    {
        pGridFieldDesc = ( PMAGRIDFIELDDESC )pGridFieldDescLL->data;
        printf("Dataset Name: %s\n",
               pGridFieldDesc->szDataSetName);
        printf("RecordID: %d\n",
               pGridFieldDesc->lRecordId);
        n++;
        pGridFieldDescLL = pGridFieldDescLL->next;
    }
    pGridFieldDesc = ( PMAGRIDFIELDDESC ) GridFieldDescLL.data;
```

```
strcpy ( GridRefData.szDataSetName,
          pGridFieldDesc->szDataSetName ) ;
GridRefData.lRecordId = pGridFieldDesc->lRecordId;
GridRefData.stGeoArea.rsNLat      = 60.0;
GridRefData.stGeoArea.rsSLat      = 15.0;
GridRefData.stGeoArea.rsWLon     = 100.0;
GridRefData.stGeoArea.rsELon     = 158.0;
GridRefData.stGridFormat.eFormat   = MAGRID_AS_SPECIFIED;
GridRefData.stGridFormat.lMaxXPoint = pGridFieldDesc->lMaxXPoint;
GridRefData.stGridFormat.lMaxYPoint = pGridFieldDesc->lMaxYPoint;
GridRefData.stGridFormat.eScanMode  = MAGRID_pXinnY;
GridRefData.stGridFormat.stProjectionDesc.eProjection = MAGRID_SPHERICAL;
GridRefData.stGridFormat.lUnitId    = 640;

/*****************************************/
/* Retrieve Grid Data by Reference. */
/*****************************************/
magridRet = MAGRIDGet2DByID ( &GridRefData, &stGridData );
if (magridRet.nStatus)
{
    printf( "Error in MAGRIDGet2DByID (%d) (%s)\n", magridRet.nStatus,
            magridRet.szSQLState );
    printf( "ErrorMessage (%s)\n", magridRet.szErrorMessage );

}
else
{
    printf( "SUCCESSFUL Retrieve of Grid DatasetName=(%s)
            RecordId(%d)\n",
            GridRefData.lRecordId, GridRefData.szDataSetName );
    /* Free the Grid data */
    free (pGridData->pGridFieldData);
}
}

/*****************************************/
/* Call MAGRIDDisconnect. */
/*****************************************/
magridRet = MAGRIDDisconnect();
}
DPRINT( "tester: Exiting (%d).\n", magridRet.nStatus );
exit( magridRet.nStatus );
}
```

4.1.8 Retrieving 3D Grid Field Data From the Database

Four methods are provided for retrieving 3D grid field data. Three of the methods, MAGRIDGetVolumeByID, MAGRIDGetSliceByID, and MAGRIDGetProfileByID, take as input a datasetname and record id of the 3D grid field to be retrieved. The datasetname and the record identifier are typically found by calling MAGRID3DCatalog to get the list of grid fields matching a specified criteria. The MAGRIDGetVolumeByID and the MAGRIDGetSliceByID also allow the user to specify the geometry of returned grid by filling in the MAGRIDREFERENCE data structure. The MAGRIDGetSliceByID also takes as input a level indicator for extracting a single horizontal grid slice from the designated 3D grid volume. The

MAGRIDGetProfileByID allows specification of a single latitude/longitude pair for the return of a single 3D grid profile (stick), via the MAGRID3DSTICK data structure. The MAGRIDGetVolumeByID returns a 3D volume in the MAGRID3DDATA structure. The MAGRIDGetSliceID returns a 2D slice in the MAGRIDDATA structure. The return structures, MAGRID3DSTICK, MAGRID3DDATA, and MAGRIDDATA, all have pointers to the data. These data pointers should be freed after use and then the structures themselves freed if allocated dynamically.

The last method for retrieving 3D grid data is MAGRIDGetTrack. This method takes as input a MAGRID3DTRACKQUERY structure that describes the track requirements (e.g., resolution, range, bearing, starting latitude/longitude point), as well as information describing grid type and time. The routine will return the best profile points found along the specified track from the grid type of interest. Best data is based on forecasts closest to current system time but no older than the end base time value specified by the user. The MAGRIDGetTrack routine returns the number of track points found that meet the specified criteria, as well as a linked list of MAGRID3DSTICK profiles. The routine MAGRIDFreeLL should be called to free the linked list after use, and then the head of the linked list may also need to be freed if allocated dynamically. All four routines return a MAGRIDRET return status structure.

The code example below shows the use of MAGRIDGetVolumeByID, MAGRIDGetVolumePtr, MAGRIDGetProfileByID, MAGRIDGetSliceByID, and MAGRIDGetTrack to retrieve 3D grid data, and it also provides an example of the use of MAGRID3DCatalog as well as the retrieve by stored capability. The MAGRIDFORMAT structure contains an enumerated type, eOutputFormat. This field may be set to MAGRID_GET_AS_SPECIFIED (default setting when memset of structure done), in which case the grid field data will be retrieved in the format specified in the GRIDREFERENCE and/or GRIDFORMAT structure. Otherwise, it may be set to MAGRID_GET_AS_STORED, and the grid field data will be retrieved as it was ingested. No subgridding, projection/unit conversion, or registration will be applied. A retrieval using the MAGRID_GET_AS_STORED flag will typically need a corresponding MAGRIDRetrRegistration call.

```
*****
T   E   S   T   E   R
*****
Purpose: Sample code using MAGRID 3D retrieve routines
          (Catalog, Volume, Profile, Track and Slice)
*****
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <time.h>
#include "MAGRIDAPI.h"
#ifndef _WIN32
#define _MDBDLL
#endif
```

```
***** M A I N *****
*****
void main(argc, argv)
    int argc;
    char **argv;
{

FILE          *fp;
FILE          *fp2;
int           n = 0;
int           i = 0;
int           j = 0;
int           x = 0;
int           y = 0;
int           z = 0;
MAGRIDRET     magridRet;
MAGRID3DQUERY GridQuery;
MAGRIDREFERENCE GridRefData;
MAGRIDDATA    stGridData;
MAGRID3DDATA   st3DGridData;
PMAGRID3DFIELDDESC pFieldDesc, pGridFieldDesc = NULL;
MAGRIDFORMAT  stGridFormat;
MAGRID3DSTICK  ProfileData, *pStickData;
MAGRID3DTRACKQUERY TrackQuery;
long          lNumFound =0;
long          lNumberTrackPts =0;
char          szGridDataFileName[ 20 ];
char          szDigit[9];
float         *pData = NULL;
MAGRIDLINKEDLIST GridDataLL, *pGridDataLL,
                   GridFieldDescLL, *pGridFieldDescLL, TrackData;
PMAGRIDLINKEDLIST pLL;
float         rsLevel;

printf("*****\n");
printf("Testing MAGRID 3D Retrieves\n");
printf("\n\n");

magridRet = MAGRIDConnect();

if (magridRet.nStatus)
{
    printf( "\tError calling MAGRIDConnect Status (%d) SQLState (%s)\n",
            magridRet.nStatus, magridRet.szSQLState );
    printf( "\tErrorMessage (%s)\n", magridRet.szErrorMessage );
    exit ( 0 );
}

else
{
    /* Populate the catalog query structure */
    ****
    memset(&GridQuery, 0, sizeof(MAGRID3DQUERY));
    GridQuery.lGeneratingProcId    = 200;           /*MODAS Data */
    GridQuery.lProductionCenterId = 58;             /* US Navy */
    GridQuery.lSubCenterId       = 0;                /* FNMOC */
    GridQuery.lGridId            = MAGRID_QUERY_WILDCARD;
    GridQuery.lParameterId      = MAGRID_QUERY_WILDCARD;
    GridQuery.lBeginBaseTime    = MAGRID_QUERY_WILDCARD;
    */
}
```

```
GridQuery.lEndBaseTime          = MAGRID_QUERY_WILDCARD;
GridQuery.lBeginTau             = 0;           /* TAU in minutes          */
GridQuery.lEndTau               = 1440;        /* TAU in minutes (24hr)   */
GridQuery.stGeoArea.rsNLat     = 90.0;
GridQuery.stGeoArea.rsSLat     = -90.0;
GridQuery.stGeoArea.rsWLon     = -180.0;
GridQuery.stGeoArea.rsELon     = 180.0;
GridQuery.eDataCategory        = MAGRID_BASE;
GridQuery.lBeginReceiptTime    = MAGRID_QUERY_WILDCARD;
GridQuery.lEndReceiptTime      = MAGRID_QUERY_WILDCARD;

/*****************************************/
/* First, get a catalog of 3D grids of interest */
/*****************************************/
magridRet = MAGRID3DCatalog ( &GridQuery, &lNumFound, &GridDataLL );
if ( magridRet.nStatus )
{
    printf( "\tError in MAGRID3DCatalog Status (%d) SQLState (%s)\n",
            magridRet.nStatus, magridRet.szSQLState );
    printf( "\tErrorMessage (%s)\n", magridRet.szErrorMessage );
}
else
{
    printf( "\tMAGRID3DCatalog : Successful!\n" );

    if ( lNumFound < 1 )
    {
        printf( "\nNo grids were found.\n" );
    }
    else
    {
       /*****************************************/
        /* Populate the MAGRIDREFERENCE structure */
       /*****************************************/
        pGridDataLL = &GridDataLL;
        pGridFieldDesc = ( PMAGRID3DFIELDDESC )pGridDataLL->data;
        strcpy ( GridRefData.szDataSetName,
                 pGridFieldDesc->szDataSetName );
        GridRefData.lRecordId           =
            pGridFieldDesc->lRecordId;

        GridRefData.stGridFormat.eOutputFormat = MAGRID_GET_AS_STORED;

       /*****************************************/
        /* Get Volume */
       /*****************************************/
        printf( "\n" );
        printf("DatasetName: %s\n", GridRefData.szDataSetName );
        printf("RecordID: %d\n", GridRefData.lRecordId );
        printf( "\n" );
        magridRet = MAGRIDGetVolumeByID ( &GridRefData, &st3DGridData );
        if ( magridRet.nStatus )
        {
            printf(
                "\tError in MAGRIDGetVolumeByID Status (%d) SQLState(%s)\n",
                magridRet.nStatus, magridRet.szSQLState );
            printf( "\tErrorMessage (%s)\n", magridRet.szErrorMessage );
        }
        else
    }
}
```

```
{  
    printf( "\tMAGRIDGetVolumeByID : Successful\n" );  
  
    /*****  
    /* Display output values. */  
    *****/  
    printf("\n");  
    printf("Dataset Ref Name: %s\n",  
          st3DGridData.stGridFieldDesc.szDataSetName);  
    printf("Record ID: %d\n",  
          st3DGridData.stGridFieldDesc.lRecordId);  
    printf("Grid ID: %d\n",  
          st3DGridData.stGridFieldDesc.lGridId);  
    printf("Base Time: %d:%s\n",  
          st3DGridData.stGridFieldDesc.lBaseTime,  
          ctime ((time_t *)  
                  &st3DGridData.stGridFieldDesc.lBaseTime));  
    printf("GeneratingProc ID: %d\n",  
          st3DGridData.stGridFieldDesc.lGeneratingProcId);  
    printf("Production Center ID: %d\n",  
          st3DGridData.stGridFieldDesc.lProductionCenterId);  
    printf("SubCenter ID: %d\n",  
          st3DGridData.stGridFieldDesc.lSubCenterId);  
    printf("Parameter ID: %d\n",  
          st3DGridData.stGridFieldDesc.lParameterId);  
    printf("Parameter Unit ID: %d\n",  
          st3DGridData.stGridFieldDesc.lParameterUnit);  
    printf("Level Type ID: %d\n",  
          st3DGridData.stGridFieldDesc.lLevelType);  
    printf("Receipt Time: %ld:%s\n",  
          st3DGridData.stGridFieldDesc.lReceiptTime,  
          ctime ((time_t *)  
                  &st3DGridData.stGridFieldDesc.lReceiptTime));  
    printf("Tau: %ld\n",st3DGridData.stGridFieldDesc.lTau);  
    printf("Quality Indicator: %ld\n",  
          st3DGridData.stGridFieldDesc.lQualityIndicator);  
    printf("Data Category: %ld\n",  
          st3DGridData.stGridFieldDesc.eDataCategory);  
    printf("Security Class: %s\n",  
          st3DGridData.stGridFieldDesc.szSecurityClass);  
    printf("Receipt Method: %s\n",  
          st3DGridData.stGridFieldDesc.szReceiptMethod);  
    printf("Compression: %s\n",  
          st3DGridData.stGridFieldDesc.szCompression);  
    printf("Projection: %d\n",  
          st3DGridData.stGridFieldDesc.eProjection);  
  
    printf("Number of Levels: %d\n",  
          st3DGridData.stGridFieldDesc.lMaxZPoint);  
    printf("Retrieved Levels:\n");  
    for (i = 0; i < st3DGridData.stGridFieldDesc.lMaxZPoint; i++)  
    {  
        printf("Level[%d]: %f\n", i,  
              st3DGridData.stGridFieldDesc.rsLevel[i]);  
    }  
  
    printf("Size: %d\n",st3DGridData.ulSize);  
  
    /* Construct file name for Grid Data output */  
    strcpy(szGridDataFileName,"GridVol_GFD");  
    sprintf(szDigit,"%d",j);
```

```
strcat(szGridDataFileName,szDigit);

fp2 = fopen(szGridDataFileName,"w");

for (z = 0; z < st3DGridData.stGridFieldDesc.lMaxZPoint; z++)
{
    /*****
    /* Get Volume Pointer */
    *****/
    magridRet = MAGRIDGetVolumePtr (st3DGridData,
                                    st3DGridData.stGridFieldDesc.rsLevel[z],
                                    &pData);
    if ( !magridRet.nStatus )
    {
        n = 0;
        fprintf(fp2,"LEVEL[%d]\n", z);
        for (y = 0; y < st3DGridData.stGridFieldDesc.lMaxYPoint;
             y++)
        {
            for (x = 0; x < st3DGridData.stGridFieldDesc.lMaxXPoint;
                 x++)
            {
                fprintf(fp2,"%f ", pData[n]);
                n++;
            }
            fprintf(fp2,"\n");
        }
    }
}

fclose(fp2);

free (stGridData.pGridFieldData);

printf("Grid Volume Data has been written to file %s\n\n",
      szGridDataFileName);
}

/*****
/* Get Slice.  Use same reference data as for Volume. */
****/
rsLevel = st3DGridData.stGridFieldDesc.rsLevel[0];
                    /* Level 0 */

magridRet = MAGRIDGetSliceByID ( &GridRefData,
                                 rsLevel, &stGridData );
if (magridRet.nStatus)
{
    printf(
        "\tError in MAGRIDGetSliceByID Status (%d) SQLState (%s)\n",
        magridRet.nStatus, magridRet.szSQLState );
    printf( "\tErrorMessage (%s)\n", magridRet.szErrorMessage );
}
else
{
    printf( "\tMAGRIDGetSliceByID : Successful\n" );

    /*****
    /* Display output values. */
    ****/
```

```
printf("\n");
printf("Level Returned: %f\n\n", rsLevel);
printf("Dataset Ref Name: %s\n",
      stGridData.stGridFieldDesc.szDataSetName);
printf("Record ID: %d\n",
      stGridData.stGridFieldDesc.lRecordId);
printf("Grid ID: %d\n",
      stGridData.stGridFieldDesc.lGridId);
printf("Base Time: %d:%s\n",
      stGridData.stGridFieldDesc.lBaseTime,
      ctime ((time_t *) &stGridData.stGridFieldDesc.lBaseTime));
printf("GeneratingProc ID: %d\n",
      stGridData.stGridFieldDesc.lGeneratingProcId);
printf("Production Center ID: %d\n",
      stGridData.stGridFieldDesc.lProductionCenterId);
printf("SubCenter ID: %d\n",
      stGridData.stGridFieldDesc.lSubCenterId);
printf("Parameter ID: %d\n",
      stGridData.stGridFieldDesc.lParameterId);
printf("Parameter Unit ID: %d\n",
      stGridData.stGridFieldDesc.lParameterUnit);
printf("Level Type ID: %d\n",
      stGridData.stGridFieldDesc.lLevelType);
printf("Receipt Time: %ld:%s\n",
      stGridData.stGridFieldDesc.lReceiptTime,
      ctime ((time_t *)
              &stGridData.stGridFieldDesc.lReceiptTime));
printf("Tau: %ld\n", stGridData.stGridFieldDesc.lTau);
printf("Lowest Level: %ld\n",
      stGridData.stGridFieldDesc.rsLevelLo);
printf("Highest Level: %ld\n",
      stGridData.stGridFieldDesc.rsLevelHi);
printf("Quality Indicator: %ld\n",
      stGridData.stGridFieldDesc.lQualityIndicator);
printf("Data Category: %ld\n",
      stGridData.stGridFieldDesc.eDataCategory);
printf("Security Class: %s\n",
      stGridData.stGridFieldDesc.szSecurityClass);
printf("Receipt Method: %s\n",
      stGridData.stGridFieldDesc.szReceiptMethod);
printf("Compression: %s\n",
      stGridData.stGridFieldDesc.szCompression);
printf("Projection: %d\n",
      stGridData.stGridFieldDesc.eProjection);
printf("Size: %d\n", stGridData.ulSize);

/* Construct file name for Grid Data output */
strcpy(szGridDataFileName,"Slice_GFD");
sprintf(szDigit,"%d",j);
strcat(szGridDataFileName,szDigit);

pData = ( float * ) stGridData.pGridFieldData;
n = 0;

fp2 = fopen(szGridDataFileName,"w");

for ( y = 0; y < stGridData.stGridFieldDesc.lMaxYPoint; y++)
{
    for ( x = 0; x < stGridData.stGridFieldDesc.lMaxXPoint; x++)
    {
        fprintf(fp2,"%f ", pData[n]);
    }
}
```

```
        n++;
    }
    fprintf(fp2, "\n");
}

fclose(fp2);

free (stGridData.pGridFieldData);

printf("Grid Slice Data has been written to file %s\n\n",
      szGridDataFileName);
}

/*************
/* Get Profile. */
/************/
ProfileData.rsLat = 40.0;
ProfileData.rsLon = 40.0;

magridRet = MAGRIDGetProfileByID ( pGridFieldDesc->szDataSetName,
                                   pGridFieldDesc->lRecordId,
                                   ProfileData.rsLat,
                                   ProfileData.rsLon,
                                   &ProfileData );

if (magridRet.nStatus)
{
    printf(
        "\tError in MAGRIDGetProfileByID Status (%d) SQLState (%s)\n",
        magridRet.nStatus, magridRet.szSQLState );
    printf( "\tErrorMessage (%s)\n", magridRet.szErrorMessage );
}
else
{
    printf( "\tMAGRIDGetProfileByID : Successful\n" );

    /*****
    /* Display output values. */
    *****/
    printf("\n");
    printf("Dataset Name: %s\n",
          ProfileData.stGridFieldDesc.szDataSetName);
    printf("Record ID: %d\n", ProfileData.stGridFieldDesc.lRecordId);
    printf("Grid ID: %d\n", ProfileData.stGridFieldDesc.lGridId);
    printf("Center ID: %d\n",
          ProfileData.stGridFieldDesc.lProductionCenterId);
    printf("SubCenter ID: %d\n",
          ProfileData.stGridFieldDesc.lSubCenterId);
    printf("Model ID: %d\n",
          ProfileData.stGridFieldDesc.lGeneratingProcId);
    printf("Parameter ID: %d\n",
          ProfileData.stGridFieldDesc.lParameterId);
    printf("Parameter Unit ID: %d\n",
          ProfileData.stGridFieldDesc.lParameterUnit);
    printf("Base Time: %d:%s\n",
          ProfileData.stGridFieldDesc.lBaseTime,
          ctime ((time_t *) &ProfileData.stGridFieldDesc.lBaseTime));
    printf("Receipt Time: %ld:%s\n",
          ProfileData.stGridFieldDesc.lReceiptTime,
          ctime((time_t *) &ProfileData.stGridFieldDesc.lReceiptTime));
}
```

```
printf("Tau: %ld\n", ProfileData.stGridFieldDesc.lTau);
printf("MaxXPoint: %ld\n",
       ProfileData.stGridFieldDesc.lMaxXPoint);
printf("MaxYPoint: %ld\n",
       ProfileData.stGridFieldDesc.lMaxYPoint);
printf("MaxZPoint: %ld\n",
       ProfileData.stGridFieldDesc.lMaxZPoint);
printf("Quality Indicator: %ld\n",
       ProfileData.stGridFieldDesc.lQualityIndicator);
printf("Data Category: %ld\n",
       ProfileData.stGridFieldDesc.eDataCategory);
printf("Security Class: %s\n",
       ProfileData.stGridFieldDesc.szSecurityClass);
printf("Receipt Method: %s\n",
       ProfileData.stGridFieldDesc.szReceiptMethod);
printf("Compression: %s\n",
       ProfileData.stGridFieldDesc.szCompression);
printf("Projection: %d\n",
       ProfileData.stGridFieldDesc.eProjection);
printf("Size: %d\n", ProfileData.ulSize);

pData = ( float * ) ProfileData.pGridFieldData;

for ( n = 0;
      n < ProfileData.stGridFieldDesc.lMaxZPoint;
      n++ )
{
    printf("Level[%d]: %f\tValue: %f\n", n,
           ProfileData.stGridFieldDesc.rsLevel[n], pData[n]);
}

printf("\n");
free(pData);

/*****************************************/
/* Populate the track query structure */
/*****************************************/
memset(&TrackQuery, 0, sizeof(MAGRID3DTRACKQUERY));
TrackQuery.lGeneratingProcId     = 200; /* MODAS */          */
TrackQuery.lProductionCenterId  = 58;  /* US Navy */        */
TrackQuery.lSubCenterId         = 0;   /* FNMO */          */
TrackQuery.lGridId              = MAGRID_QUERY_WILDCARD;
TrackQuery.lParameterId         = MAGRID_QUERY_WILDCARD;
TrackQuery.lBeginBaseTime      = MAGRID_QUERY_WILDCARD;
TrackQuery.lEndBaseTime        = MAGRID_QUERY_WILDCARD;
TrackQuery.rsRange              = 1000.0; /* nautical miles */
TrackQuery.rsBearing            = 90.0;  /* degrees */      */
TrackQuery.rsLat                = 60.0;
TrackQuery.rsLon                = 40.0;
TrackQuery.nResolution          = 5;      /* 5 Track Points */

/*****************************************/
/* Get Track. */
/*****************************************/
magridRet = MAGRIDGetTrack (TrackQuery, &lNumberTrackPts,
                             &TrackData);
if (magridRet.nStatus)
{
    printf("\tError in MAGRIDGetTrack (%d) (%s)\n",
           magridRet.nStatus, magridRet.szSQLState);
```

```
    printf( "\tErrorMessage (%s)\n", magridRet.szErrorMessage);
}
else
{
    printf("\tMAGRIDGetTrack : Successful\n\n");

    /* Construct file name for Data output */
    strcpy(szGridDataFileName,"Track_GFD");
    sprintf(szDigit,"%d",j);
    strcat(szGridDataFileName,szDigit);

    fp2 = fopen(szGridDataFileName,"w");

    pLL = &TrackData;

    for (i=0; i <lNumberTrackPts; i++)
    {
        pStickData = (PMAGRID3DSTICK)pLL ->data;
        pFieldDesc =
            (PMAGRID3DFIELDDESC)&pStickData->stGridFieldDesc;

        fprintf(fp2, "Track Point: %d Lat %4.2f Lon %4.2f\n",
                i, pStickData->rsLat, pStickData->rsLon);
        fprintf(fp2, "\tCenter ID: %d SubCenter ID: %d \n",
                pFieldDesc->lProductionCenterId ,
                pFieldDesc->lSubCenterId);
        fprintf(fp2, "\tlGridId: %d lGeneratingProcId %d \n",
                pFieldDesc->lGridId, pFieldDesc->lGeneratingProcId);
        fprintf(fp2, "\tlBaseTime: %d:%s",
                pFieldDesc->lBaseTime,
                ctime((time_t *) &pFieldDesc->lBaseTime));
        fprintf(fp2, "\tTau: %d\n", pFieldDesc->lTau);
        fprintf(fp2, "\tReceiptTime: %d:%s",
                pFieldDesc->lReceiptTime,
                ctime((time_t *) &pFieldDesc->lReceiptTime));
        fprintf(fp2, "\tlParameterId: %d lParameterUnit: %d \n",
                pFieldDesc->lParameterId, pFieldDesc->lParameterUnit);
        fprintf(fp2,
                "\tMaxXPoint: %d MaxYPoint: %d MaxZPoint: %d\n",
                pFieldDesc->lMaxXPoint, pFieldDesc->lMaxYPoint,
                pFieldDesc->lMaxZPoint);
        fprintf(fp2, "\tQuality Indicator: %d DataCategory: %d\n",
                pFieldDesc->lQualityIndicator,
                pFieldDesc->eDataCategory);
        fprintf(fp2, "\tszSecurityClass: %s\n",
                pFieldDesc->szSecurityClass);
        fprintf(fp2, "\tszReceiptMethod: %s\n",
                pFieldDesc->szReceiptMethod);
        fprintf(fp2, "\tszCompression: %s\n",
                pFieldDesc->szCompression);
        fprintf(fp, "\tLevelType: %d\n",
                pFieldDesc->lLevelType);

        pData = (float *)pStickData->pGridFieldData;
        for (j=0; j<pFieldDesc->lMaxZPoint; j++)
        {
            printf(fp2," \t\tLevel %7.2f Value %7.2f\n",
                   pFieldDesc->rsLevel[j], pData[j]);
        }

        pLL = pLL ->next;
    }
}
```

```
        }
        fclose(fp2);
        printf("Track Data has been written to file %s\n\n",
               szGridDataFileName);

    }
}

MAGRIDFreeLL( &GridDataLL );

/*****************/
/* Call MAGRID_stop. */
/*****************/
magridRet = MAGRIDDisconnect();
}

printf("Testing Complete\n");
printf("*****\n");

DPRINT( "tester: Exiting (%d).\n", magridRet.nStatus );
exit( magridRet.nStatus );
}
```

4.1.9 Ingesting/Retrieving a 3D EOF

The MAGRID3DIgest routine may be used to ingest a 3D EOF grid into the database. The EOF grid is treated as a Binary Large Object (BLOB) of data and therefore cannot be subgridded. The 3D grid is not reconstructed. The p3DIgest->szCompression must be set to “MAGRID_EOF_COMPRESSION.” The EOF grid can only be retrieved if the pReference->stGridFormat.eOutputFormat is set to “MAGRID_GET_AS_STORED.”

The MAGRID3DINGEST structure should be initialized to 0. The p3DIgest->rsLevel does not have to be set because it will not be used. All other fields in the MAGRID3DINGEST structure will be validated. The code below provides an example of registering/ingesting a 3D EOF grid into the database and then retrieving it from the database.

```
*****
*          T   E   S   T   E   R
/*
 * Purpose: Sample Code Ingesting and retrieving a 3D EOF file.
 *
 * Typically the EOF messages are written to a file and prefaced
 * with an OTHT Gold header. The header contains the information
 * that is needed for the registration and the metadata that is
 * stored with the grid. The EOF portion should not have to be
 * decoded/reconstructed in order to ingest the grid.
 *
 * Below is an example of an EOF message file and the corresponding
 * registration information:
 *
 *      METXCR3 98121700 XXXX 0000.ATN
 *      MSGID/FNOC/GRIDFLD/0000/DEC
 *      PROD/OCEANMET/170231Z4/DEC/000/10F1/CR3 98121700 XXXX 0000.ATB
 *      CMPCT/EOF1
```

```
/*
 * GRID/5800N3/10600W7/0M/078/056/34/1.00DEG/1.00DEG/VAR/03
 * NARR/FNMOC CDS OTIS_GLOBAL sea_temp global_360x181
 * ...EOF ...
 *
 */
/*
 * GridId          255           (MAGRID_UNDEFINED_GRIDID)
 * lProductionCenterId   58
 * lSubCenterId      0
 * lGeneratingProcId 43
 * rsRegLat         58.00
 * rsRegLon         -106.00
 * rsRegX            1
 * rsRegY            1
 * rsXDistance      111.137     (1.0 Deg * 111.137 = xDistance km)
 * rsYDistance      111.137     (1.0 Deg * 111.137 = yDistance km)
 * lMaxXPoint       78
 * lMaxYPoint       56
 * lMaxZPoint       34
 * eScanMode        MAGRID_pXinnY
 *
 * stProjectionDesc.eProjection           MAGRID_SPHERICAL;
 * stProjectionDesc.projParms.spherical.rsXResolution 1           (1.0 Deg)
 * stProjectionDesc.projParms.spherical.rsYResolution 1           (1.0 Deg)
 * szRegName          " "
 * szModelName        " "
 * szCenterName       " "
 *
 * GridData.lParameterId      = 11      ( T from 0000.ATN indicates temperature)
 * GridData.lLevelType        = 160     ( A from 0000.ATN indicates LevelType )
 */
*/
*****
*****include <stdlib.h>
*****include <stdio.h>
*****include <string.h>
*****include <time.h>
*****include "MAGRIDAPI.h"

#define MINUTES_PER_HOUR 60

*****
*****          M   A   I   N
*****/
void main(argc, argv)
    int argc;
    char **argv;
{
    struct tm      timeptr;
    char          *pData;
    MAGRIDREFERENCE GridRefData;
    MAGRID3DDATA   st3DGridData;
    int           index = 0;
    int           lProductionCenterId, lSubCenterId, lGeneratingProcId;
    MAGRIDRET     magridRet;
    MAGRID3DINGEST GridData;
    MAGRIDINPUTREG RegData;
    PMAGRIDSUPERICAL pSphericalData;

    magridRet = MAGRIDConnect();
    if (magridRet.nStatus)
```

```
{  
    printf( "\ttester:Error calling MAGRIDConnect Status (%d) SQLState(%s)\n",
            magridRet.nStatus, magridRet.szSQLState );
    printf( "\tErrorMessage (%s)\n", magridRet.szErrorMessage );
}  
else  
{  
  
    /*****  
    /* Register the MODAS 3-D Grid with the      */  
    /* database.                                     */  
    /*****  
    lProductionCenterId = 58;  
    lSubCenterId       = 0;  
    lGeneratingProcId = 43;  
  
    /* Fill in Registration information for 3D example */  
    memset(&RegData, 0, sizeof(MAGRIDINPUTREG));  
    RegData.lGridId = MAGRID_UNDEFINED_GRIDID;  
    RegData.lProductionCenterId = lProductionCenterId;  
    RegData.lSubCenterId = lSubCenterId;  
    RegData.lGeneratingProcId = lGeneratingProcId;  
  
    RegData.rsRegLat = 58.0;  
    RegData.rsRegLon = -106.0;  
    RegData.rsRegX   = 1.0;  
    RegData.rsRegY   = 1.0;  
    RegData.rsXDistance = 111.137;  
    RegData.rsYDistance = 111.137;  
    RegData.lMaxXPoint = 78;  
    RegData.lMaxYPoint = 56;  
    RegData.lMaxZPoint = 34;  
    RegData.eScanMode = MAGRID_pXinnY;  
  
    RegData.stProjectionDesc.eProjection = MAGRID_SPHERICAL;  
    pSphericalData = (PMAGRIDSPHERICAL) &RegData.stProjectionDesc.projParms;  
    pSphericalData->rsXResolution = 1.0;  
    pSphericalData->rsYResolution = 1.0;  
  
    /*****  
    /* Call MAGRIDRegisterModel. */  
    /*****  
    magridRet = MAGRIDRegisterModel( &RegData ) ;  
  
    if (magridRet.nStatus)  
    {  
        printf( "\tMAGRIDRegisterModel: Error (%d)!!\n",
                magridRet.nStatus );  
    }  
    else  
    {  
        printf("\tMAGRIDRegisterModel: Successful!!\n");
        printf("\t\tGridID of registration is %d\n", RegData.lGridId);  
  
        memset(&GridData, 0, sizeof(MAGRID3DINGEST));
        GridData.lGeneratingProcId = lGeneratingProcId;
        GridData.lGridId           = RegData.lGridId;
        GridData.lProductionCenterId = lProductionCenterId;
        GridData.lSubCenterId       = lSubCenterId;
        GridData.lParameterId      = 11; /* Temp */
        GridData.lParameterUnit     = 0; /* Use Default */
```

```
GridData.lLevelType      = 160;
GridData.lMaxZPoint     = RegData.lMaxZPoint;

/* Compute basetime (epoch) from Jan 1, 1998 */
memset(&timeptr,0,sizeof(struct tm));
timeptr.tm_year = 98;
timeptr.tm_mon = 1;
timeptr.tm_mday = 1;
timeptr.tm_hour = 0;
GridData.lBaseTime      = (long) mktime(&timeptr);

/* 24 hour forecast */
GridData.lTau            = 24 * MINUTES_PER_HOUR;
GridData.lQualityIndicator = 0;
GridData.eDataCategory   = MAGRID_BASE;
strcpy(GridData.szSecurityClass, "UNCLASS");
strcpy(GridData.szReceiptMethod, "NET");
strcpy(GridData.szCompression, "MAGRID_EOF_COMPRESSION");
strcpy(GridData.szDescription, "DummyData");

/* Set the ulSize to the size of the Message File */
GridData.ulSize = 57567;

pData = (char *) malloc(GridData.ulSize);

/*****************/
/* Populate a dummy grid and fill in level data */
/*****************/
for ( index = 0; index < GridData.ulSize; index++)
{
    pData[index] = index;
}

GridData.pGridFieldData = (void *) pData;

/*****************/
/* Call MAGRID3DIgest. */
/*****************/
magridRet = MAGRID3DIgest( &GridData ) ;

if (magridRet.nStatus)
{
    printf( "\tMAGRID3DIgest: Error! Status (%d) SQLState (%s)\n",
           magridRet.nStatus, magridRet.szSQLState );
    printf( "\tErrorMessage (%s)\n", magridRet.szErrorMessage );

}
else
{
    printf( "\tMAGRID3DIgest: Successful (%s:%d)!\n",
           GridData.szDataSetName, GridData.lRecordId );
}
free(pData);
}

/*****************/
/* Now Retrieve the EOF Data */
/* Must retrieve is AS STORED */
/*****************/

if (!magridRet.nStatus)
```

```
{  
    memset(&GridRefData, 0, sizeof(MAGRIDREFERENCE));  
    GridRefData.lRecordId      = GridData.lRecordId;  
    strcpy (GridRefData.szDataSetName, GridData.szDataSetName);  
    GridRefData.stGridFormat.eOutputFormat = MAGRID_GET_AS_STORED;  
    magridRet = MAGRIDGetVolumeByID ( &GridRefData, &st3DGridData );  
  
    if (magridRet.nStatus)  
    {  
        printf( "Error in MAGRID3DGetVolumeByID (%d)\n",  
                magridRet.nStatus );  
        printf( "\t\t(%s:%s)\n", magridRet.szSQLState,  
                magridRet.szErrorMessage );  
    }  
    else  
    {  
        printf ("Successful Retrieval of EOF Grid\n");  
    }  
}  
  
/****************************************/  
/* Call MAGRID_stop. */  
/****************************************/  
magridRet = MAGRIDDisconnect();  
  
}  
  
exit( magridRet.nStatus );  
}
```

4.1.10 Updating Grid Field Points in the Database

The MAGRIDUpdateByID method is used to update 2D and 3D grid field points contained in the database. This method requires the dataset name and the record id for the grid field to be updated. Typically, the MAGRIDGet2DByID, MAGRIDGet2DByQuery, or MAGRIDGetVolume method is used to get the grid field data for update. MAGRIDGet2DByQuery is shown in the example below. Note that the **MAGRID_GET_AS_STORED** field is set when the retrieval is done. This will need to be done to correctly update the grid field points (should not update grids based on interpolated or projected points). Once the grid of interest is extracted and then the grid field points updated, the MAGRIDUpdateByID call may be made by passing in the reference identifiers as well as the size and actual updated grid points. An update for a grid where eDataCategory is set to **MAGRID_BASE** grid field is stored as a new grid field record. Grids modified by the MAGRIDUpdateByID field will have their **eDataCategory** field set to **MAGRID_EDITED**.

```
*****  
T E S T E R  
*****  
Purpose: Sample code using MAGRIDUpdateByID.  
*****  
#include <stdlib.h>  
#include <stdio.h>
```

```
#include "MAGRIDAPI.h"

#define SECONDS_PER_HOUR 60*60
#ifndef _WIN32
#define _MDBDLL
#endif

void main(argc, argv)
    int argc;
    char **argv;
{
    MAGRIDLINKEDLIST      GridDataLL;
    MAGRIDFORMAT          GridFormat;
    MAGRIDQUERY           GridQuery;
    PMAGRIDDATA          pLL;
    MAGRIDRET             magridRet;

    long                  lNumFound;
    float                 *pGridData;

    magridRet = MAGRIDConnect();

    if (magridRet.nStatus)
    {
        printf( "\tError calling MAGRIDConnect (%d) (%s)\n",
                magridRet.nStatus, magridRet.szSQLState );
        printf( "\tErrorMessage (%s)\n", magridRet.szErrorMessage );
    }
    else
    {
        printf( "\tMAGRIDConnect : Successful!\n");

        /***** Set up the Query to get the grid of interest ****/
        /* Set up the Query to get the grid of interest */
        /***** */

        memset(&GridQuery, 0, sizeof(MAGRIDQUERY));
        GridQuery.lProductionCenterId   = 58;           /* US Navy */           */
        GridQuery.lSubCenterId         = 0;            /* FNMOC */            */

        /* NOGAPS model*/
        GridQuery.lGeneratingProcId   = 58;
        GridQuery.lGridId              = 223;

        /* Sea Surface Temperature */
        GridQuery.lParameterId        = 201;

        /* Last Generated Grid Set for 0 hour forecast */
        GridQuery.lBeginBaseTime      = time(0) - 12*SECONDS_PER_HOUR;
        GridQuery.lBeginBaseTime      = time(0);

        GridQuery.lBeginTau            = 0;             /* TAU in minutes */
        GridQuery.lEndTau              = 0;             /* (0hr) */

        GridQuery.rsBeginLevel        = 0.0;
        GridQuery.rsEndLevel          = 0.0;           /* Used if Layer Type */
        GridQuery.lLevelType          = 100;           /* isobaric level */
        GridQuery.StGeoArea.rsNLat    = 90;
        GridQuery.StGeoArea.rsSLat    = -90
    }
}
```

```
GridQuery. StGeoArea.rsWLon      = -180
GridQuery. StGeoArea.rsELon      = 180
GridQuery.eDataCategory        = MAGRID_BASE;
GridQuery.lBeginReceiptTime    = MAGRID_QUERY_WILDCARD;
GridQuery.lEndReceiptTime      = MAGRID_QUERY_WILDCARD;
GridFormat.eOutputFormat       = MAGRID_GET_AS_STORED;

/*****************************************/
/* Retrieve Grid of Interest. */
/*****************************************/
magridRet = MAGRIDGet2DByQuery ( &GridQuery, &GridFormat,
                                &lNumFound, &GridDataLL );

if (magridRet.nStatus)
{
    printf( "Error in MAGRIDGet2DByQuery (%d) (%s)\n",
            magridRet.nStatus, magridRet.szSQLState );
}
else
{
    printf( "\tMAGRIDGet2DByQuery : Successful!\n" );

/*****************************************/
/* Display output values. */
/*****************************************/
if (lNumFound < 1)
{
    printf("\nNo grids were found.\n");
}
else
{
    pLL = ( PMAGRIDDATA ) GridDataLL.data;

    printf("\t\tDataset Ref Name: %s\n",
           pLL->stGridFieldDesc.szDataSetName);
    printf("\t\tRecord ID: %d\n",pLL->stGridFieldDesc.lRecordId);

/*****************************************/
/* Update the Grid Field Data. */
/*****************************************/

pGridData = (float *)pLL->pGridFieldData;

pGridData[200] = 24.8; /* Make a correction
                      to a specific grid
                      point */

magridRet = MAGRIDUpdateByID ( pLL->stGridFieldDesc.szDataSetName,
                               &pLL->stGridFieldDesc.lRecordId,
                               pLL->ulSize, pGridData );

if (magridRet.nStatus)
{
    printf("Unable to Update: %s RecordID %d (error:%d)\n",
           pLL->stGridFieldDesc.szDataSetName,
           pLL->stGridFieldDesc.lRecordId,
           magridRet.nStatus);
    printf( "ErrorMessage (%s)\n", magridRet.szErrorMessage );
}
else
```

```
{  
    printf("\tMAGRIDUpdateByID Success: %s RecordID %d\n",  
          pLL->stGridFieldDesc.szDataSetName,  
          pLL->stGridFieldDesc.lRecordId);  
  
}  
}  
/* ***** */  
/* Call MAGRID_stop. */  
/* ***** */  
magridRet = MAGRIDDisconnect();  
  
}  
printf( "Exiting (%d).\n", magridRet.nStatus );  
exit( magridRet.nStatus );  
}
```

4.1.11 Getting a 2DCatalog of Grid Fields From the Database

The MAGRID2DCatalog method is used to retrieve a catalog of 2D grid fields in the database that meet specific criteria. The criteria are contained in the input MAGRIDQUERY structure. Most fields in this structure are optional; the only requirements are for the MAGRIDGEOAREA element to be filled in with the area of interest boundaries and that a wildcard setting on parameter id requires an explicit setting for center and subcenter ids. These may be set to global (latitude -90.0 to 90.0, longitude -180.0 to 180.0) if no specific area is desired. MAGRID2DCatalog returns a linked list of MAGRIDLINKEDLIST structures, which in turn point to the metadata for each record found that met the input criteria. It also returns the number of matching records found and the MAGRIDRET status structure.

The linked list returned by MAGRID2DCatalog must be freed with a call to MAGRIDFreeLL when no longer needed. See Section 4.1.5 for an example showing the use of the MAGRID2DCatalog method.

4.1.12 Getting a 3DCatalog of Grid Fields From the Database

The MAGRID3DCatalog method is used to retrieve a catalog of 3D grid fields in the database that meet specific criteria. The criteria are contained in the input MAGRID3DQUERY structure. Most fields in this structure are optional; the only requirements are for the MAGRIDGEOAREA element to be filled in with the area of interest boundaries and that a wildcard setting on parameter id requires an explicit setting for center and subcenter ids. These may be set to global (latitude -90.0 to 90.0, longitude -180.0 to 180.0) if no specific area is desired. MAGRID3DCatalog returns a linked list of MAGRIDLINKEDLIST structures, which in turn point to the metadata for each record found that met the input criteria. It also returns the number of matching records found and the MAGRIDRET status structure.

The linked list returned by MAGRID3DCatalog must be freed with a call to MAGRIDFreeLL when no longer needed. See Section 4.1.8 for an example showing the use of the MAGRID3DCatalog method.

4.1.13 Retrieving Registration(s) From the Database

The MAGRIDRetrRegistration method is used to retrieve registration information from the database that meets specific criteria. The criteria are grid id, center id, and subcenter id, any of which may be wildcarded. MAGRIDRetrRegistration returns a linked list of MAGRIDLINKEDLIST structures, which in turn point to MAGRIDOUTPUTREG structures. It also returns the MAGRIDRET status structure. The routine MAGRIDFreeLL should be called to free the linked list after use, and then the head of the linked list may also need to be freed if allocated dynamically.

```
***** T E S T E R *****  
Purpose: Sample code for MAGRIDRetrRegistration.  
***** /  
#include <stdio.h>  
#include "MAGRIDAPI.h"  
#ifdef _WIN32  
#define _MDBDLL  
#endif  
  
void main(argc, argv)  
    int argc;  
    char **argv;  
{  
    MAGRIDLINKEDLIST    RegistrationLL,  
        *pLoopReg = NULL;  
    PMGRIDOUTPUTREG    pLL;  
    long                lGridId;  
    long                lCenterId;  
    long                lSubCenterId;  
    MAGRIDRET          magridRet;  
  
    magridRet = MAGRIDConnect();  
  
    if (magridRet.nStatus)  
    {  
        printf( "\tError calling MAGRIDConnect (%d) (%s)\n",  
            magridRet.nStatus, magridRet.szSQLState );  
        printf( "ErrorMessage (%s)\n", magridRet.szErrorMessage );  
    }  
    else  
    {  
        printf( "\tMAGRIDConnect: Successful!\n" );  
        /* Request all registrations from US Navy  
         - Fleet Numerical Oceanography Center */  
        lGridId = MAGRID_QUERY_WILDCARD;
```

```
lCenterId = 58;
lSubCenterId = 0;

/*****************/
/* Initialize linked list */
/*****************/
RegistrationLL.prev = RegistrationLL.next =
    RegistrationLL.data = ( void * ) NULL;

/*****************/
/* Retrieve Registration. */
/*****************/
magridRet = MAGRIDRetrRegistration ( lGridId, lCenterId, lSubCenterId,
    &RegistrationLL );

if (magridRet.nStatus)
{
    printf( "\tError in MAGRIDRetrRegistration (%d) (%s)\n",
            magridRet.nStatus, magridRet.szSQLState );
}
else
{
    printf( "\tMAGRIDRetrReg: Successful!\n" );

   /*****************/
    /* Display output values. */
   /*****************/
    if (RegistrationLL.data == NULL)
    {
        printf("\nNo matching registrations were found.\n");
    }
    else
    {
        for ( pLoopReg = &RegistrationLL;
              pLoopReg && RegistrationLL.data;
              pLoopReg = ( MAGRIDLINKEDLIST * ) pLoopReg->next )
        {
            PLL = ( PMGRIDOUTPUTREG ) pLoopReg->data;

            printf("\n");
            printf("Grid ID: %d\n",PLL->lGridId);
            printf("Production Center ID: %d\n",
                   PLL->lCenterId);
            printf("SubCenter ID: %d\n",PLL->lSubCenterId);
            printf("Registered Latitude: %f\n",PLL->rsRegLat);
            printf("AOIID: %d/n",PLL->lAoIID);
            printf("North Latitude: %f/n",PLL->stGeoArea.rsNLat);
            printf("South Latitude: %f/n",PLL->stGeoArea.rsSLat);
            printf("West Longitude: %f/n",PLL->stGeoArea.rsWLon);
            printf("East Longitude: %f/n",PLL->stGeoArea.rsELon);
            printf("Registered Longitude: %f\n",PLL->rsRegLon);
            printf("Registered X Coordinate: %f\n",PLL->rsRegX);
            printf("Registered Y Coordinate: %f\n",PLL->rsRegY);
            printf("Max X Point: %ld\n",PLL->lMaxXPoint);
            printf("Max Y Point: %ld\n",PLL->lMaxYPoint);
            printf("Max Z Point: %ld\n",PLL->lMaxZPoint);
            printf ("Xdistance: %f\n", PLL->rsXDistance);
            printf ("Ydistance: %f\n", PLL->rsYDistance);
            printf("Scan Mode: %d\n",PLL->eScanMode);
            printf("Projection: %d\n", PLL->stProjectionDesc.eProjection);
```

```
switch(pLL->stProjectionDesc.eProjection)
{
    case MAGRID_POLAR:
        printf("Standard Latitude: %f\n",
               pLL->stProjectionDesc.projParms.polarStereo.rsStandardLat);
        printf("Standard Longitude: %f\n",
               pLL->stProjectionDesc.projParms.polarStereo.rsStandardLon);
        break;
    case MAGRID_LAMBERT:
        printf("Standard Latitude1: %f\n",
               pLL->stProjectionDesc.projParms.lamberConf.rsStandardLat1);
        printf("Standard Latitude2: %f\n",
               pLL->stProjectionDesc.projParms.lamberConf.rsStandardLat2);
        printf("Standard Longitude: %f\n",
               pLL->stProjectionDesc.projParms.lamberConf.rsStandardLon);
        break;
    case MAGRID_MERCATOR:
        printf("Standard Latitude1: %f\n",
               pLL->stProjectionDesc.projParms.mercator.rsStandardLat1);
        printf("Standard Latitude2: %f\n",
               pLL->stProjectionDesc.projParms.mercator.rsStandardLat2);
        printf("Standard Longitude: %f\n",
               pLL->stProjectionDesc.projParms.mercator.rsStandardLon);
        break;
    case MAGRID_SPHERICAL_PROJ:
        printf("X Resolution: %f\n",
               pLL->stProjectionDesc.projParms.spherical.rsXResolution);
        printf("Y Resolution: %f\n",
               pLL->stProjectionDesc.projParms.spherical.rsYResolution);
        break;
    } /* end switch */
    printf("Name of Registration: %s\n",pLL->szRegName);

} /* end for loop */
}
}

MAGRIDFreeLL( &RegistrationLL );

magridRet = MAGRIDDisconnect();
}
DPRINT( "tester: Exiting (%d).\n", magridRet.nStatus );
exit( magridRet.nStatus );

} /* End of main. */
```

4.1.14 Getting a Single Point From a Given 2D Grid Field

The MAGRIDGetPoint method is used to retrieve a single point of data from a given 2D grid. It takes as input a grid that was retrieved from the database where no registration was applied (MAGRID_GET_AS_STORED). It will apply the registration and return the point of interest.

The following example demonstrates the use of several MAGRID APIs. First, a grid is retrieved using MAGRIDGetByQuery, and no registration is applied. Next, the associated registration is retrieved. Finally, MAGRIDGetPoint is called two times to get grid values at two different locations.

```
*****
T   E   S   T   E   R
*****
Purpose: Sample code for MAGRID routines (GetByQuery, GetPoint)
*****
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <time.h>
#include "MAGRIDAPI.h"
*****
M   A   I   N
*****
void main(argc, argv)
    int argc;
    char **argv;
{
    MAGRIDRET      magridRet;
    MAGRIDQUERY    GridQuery;
    PMAGRIIDATA    pGridData;
    MAGRIDFORMAT   stGridFormat;
    long           lNumFound =0;
    MAGRIDLINKEDLIST GridDataLL, RegistrationLL;
    float          rsPointValue;
    PMAGRIDOUTPUTREG pReg;

    printf("*****\n");
    printf("Testing MAGRID GetPoint\n\n");

    magridRet = MAGRIDConnect();

    if (magridRet.nStatus)
    {
        printf( "\tError calling MAGRIDConnect Status (%d) SQLState (%s)\n",
               magridRet.nStatus, magridRet.szSQLState );
        printf( "\tErrorMessage (%s)\n", magridRet.szErrorMessage );
        exit ( 0 );
    }
    else
    {
        ****
        /* Populate the catalog query structure */
        ****
    }
}
```

```
memset(&GridQuery, 0, sizeof(MAGRIDQUERY));
GridQuery.lGeneratingProcId    = 75;           /* NORAPS */          */
GridQuery.lProductionCenterId = 58;            /* US Navy */          */
GridQuery.lSubCenterId        = 0;              /* FNMOC */           */
GridQuery.lGridId             = 237;            /* SoCal */            */
GridQuery.lParameterId        = MAGRID_QUERY_WILDCARD;
GridQuery.lBeginBaseTime      = MAGRID_QUERY_WILDCARD;
GridQuery.lEndBaseTime        = MAGRID_QUERY_WILDCARD;
GridQuery.lBeginTau           = 0;               /* TAU in minutes */   */
GridQuery.lEndTau             = 1440;            /* TAU in minutes (24hr) */
GridQuery.rsBeginLevel        = MAGRID_QUERY_WILDCARD;
GridQuery.rsEndLevel          = MAGRID_QUERY_WILDCARD;
GridQuery.lLevelType          = MAGRID_QUERY_WILDCARD;
GridQuery.eDataCategory       = MAGRID_GET_ALL;
GridQuery.lBeginReceiptTime   = MAGRID_QUERY_WILDCARD;
GridQuery.lEndReceiptTime     = MAGRID_QUERY_WILDCARD;
GridQuery.stGeoArea.rsNLat    = 39.0;
GridQuery.stGeoArea.rsSLat    = 29.0;
GridQuery.stGeoArea.rsWLon    = -126.5;
GridQuery.stGeoArea.rsELon    = -114.5;

/*****************************************/
/* Populate the grid format structure */
/*****************************************/
stGridFormat.eOutputFormat     = MAGRID_GET_AS_STORED;

/*****************************************/
/* Call the API MAGRIDRetr to retrieve multiple Grids */
/*****************************************/
magridRet = MAGRIDGet2DByQuery ( &GridQuery, &stGridFormat,
                                &lNumFound, &GridDataLL );

if ( magridRet.nStatus )
{
    printf( "Error in MAGRIDGet2DByQuery Status (%d) SQLState (%s)\n",
            magridRet.nStatus, magridRet.szSQLState );
    printf( "\tErrorMessage (%s)\n", magridRet.szErrorMessage );
}
else
{
    printf( "\tMAGRIDGet2DByQuery : Successful!\n" );

/*****************************************/
/* Display output values. */
/*****************************************/
if ( lNumFound < 1 )
{
    printf( "\nNo grids were found.\n" );
}
else
{
    pGridData = ( PMAGRIDDATA ) GridDataLL.data;

/*****************************************/
/* Initialize linked list */
/*****************************************/
RegistrationLL.prev = RegistrationLL.next =
    RegistrationLL.data = ( void * ) NULL;

/*****************************************/
```

```
/* Retrieve Registration. */
/*********/
magridRet = MAGRIDRetrRegistration (
    pGridData->stGridFieldDesc.lGridId,
    pGridData->stGridFieldDesc.lProductionCenterId,
    pGridData->stGridFieldDesc.lSubCenterId,
    &RegistrationLL);

if (!magridRet.nStatus)
{
    pReg = (PMAGRIDOUTPUTREG) RegistrationLL.data;

    magridRet = MAGRIDGetPoint ( 38.0, -124.0, pGridData, pReg,
                                &rsPointValue);
    DPRINT ("The Grid value at location 38N,124W is %f\n",
            rsPointValue);

    magridRet = MAGRIDGetPoint ( 30.0, -120.0, pGridData, pReg,
                                &rsPointValue);
    DPRINT ("The Grid value at location 38N,124W is %f\n",
            rsPointValue);
}
MAGRIDFreeLL( &GridDataLL );
MAGRIDFreeLL( &RegistrationLL );
}

/*********/
/* Call MAGRID_stop. */
/*********/
magridRet = MAGRIDDisconnect();

printf("Testing Complete                                         \n");
printf("*****\n");

exit( magridRet.nStatus );
}
```

4.2 MDGRID Database Reference Tables

4.2.1 MDGRID Units Table

The MDGRID Units table contains the identifiers that may be used to specify a unit type request for a grid of interest. The programmer must reference this table when setting the IParameterUnit field in the MAGRIDFORMAT, MAGRID2DINGEST, and MAGRID3DINGEST structure.

Unit ID	Unit Abbreviation	Unit Name
10	-4mb D-Val(m)	-4mb delta value
11	-4mb D-Val(cm)	-4mb D-Val(cm)
20	1/s**2	per second squared
30	1/s	per second
31	(1/s)e5	1/s * 10 to the 5th
32	1/s/m	per second per meter
40	1000mb D-Val(m)	1000mb delta value
41	1000mb D-Val(cm)	1000mb D-Val(cm)
50	100mb D-Val(m)	100mb delta value
51	100mb D-Val(cm)	100mb D-Val(cm)
60	1013.2mb D-Val(m)	1013.2 millibars delta value
61	1013.2mb D-Val(cm)	1013.2 millibars D-Val(cm)
70	1050mb D-Val(m)	1050 millibars delta value
71	1050mb D-Val(cm)	1050 millibars D-Val(cm)
80	10mb D-Val(m)	10 millibars delta value
81	10mb D-Val(cm)	10 millibars D-Val(cm)
90	1100mb D-Val(m)	1100 millibars delta value
91	1100mb D-Val(cm)	1100 millibars D-Val(cm)
100	150mb D-Val(m)	150 millibars delta value
101	150mb D-Val(cm)	150 millibars D-Val(cm)
110	1mb D-Val(m)	1 millibars delta value
111	1mb D-Val(cm)	1 millibars D-Val(cm)
120	20mb D-Val(m)	20 millibars delta value
121	20mb D-Val(cm)	20 millibars D-Val(cm)

Unit ID	Unit Abbreviation	Unit Name
130	200mb D-Val(m)	200 millibars delta value
131	200mb D-Val(cm)	200 millibars D-Val(cm)
140	226.3mb D-Val(m)	226.3 millibars delta value
141	226.3mb D-Val(cm)	226.3 millibars D-Val(cm)
150	250mb D-Val(m)	250 millibars delta value
151	250mb D-Val(cm)	250 millibars D-Val(cm)
160	25mb D-Val(m)	25 millibars delta value
161	25mb D-Val(cm)	25 millibars D-Val(cm)
170	2mb D-Val(m)	2 millibars delta value
171	2mb D-Val(cm)	2 millibars D-Val(cm)
180	300mb D-Val(m)	300 millibars delta value
181	300mb D-Val(cm)	300 millibars D-Val(cm)
190	30mb D-Val(m)	30 millibars delta value
191	30mb D-Val(cm)	30 millibars D-Val(cm)
200	350mb D-Val(m)	350 millibars delta value
201	350mb D-Val(cm)	350 millibars D-Val(cm)
210	400mb D-Val(m)	400 millibars delta value
211	400mb D-Val(cm)	400 millibars D-Val(cm)
220	450mb D-Val(m)	450 millibars delta value
221	450mb D-Val(cm)	450 millibars D-Val(cm)
230	500mb D-Val(m)	500 millibars delta value
231	500mb D-Val(cm)	500 millibars D-Val(cm)
240	50mb D-Val(m)	50 millibars delta value
241	50mb D-Val(cm)	50 millibars D-Val(cm)
250	550mb D-Val(m)	550 millibars delta value
251	550mb D-Val(cm)	550 millibars D-Val(cm)
260	5mb D-Val(m)	5 millibars delta value
261	5mb D-Val(cm)	5 millibars D-Val(cm)
270	600mb D-Val(m)	600 millibars delta value
271	600mb D-Val(cm)	600 millibars D-Val(cm)
280	650mb D-Val(m)	650 millibars delta value

Unit ID	Unit Abbreviation	Unit Name
281	650mb D-Val(cm)	650 millibars D-Val(cm)
290	700mb D-Val(m)	700 millibars delta value
291	700mb D-Val(cm)	700 millibars D-Val(cm)
300	70mb D-Val(m)	70 millibars delta value
301	70mb D-Val(cm)	70 millibars D-Val(cm)
310	750mb D-Val(m)	750 millibars delta value
311	750mb D-Val(cm)	750 millibars D-Val(cm)
320	800mb D-Val(m)	800 millibars delta value
321	800mb D-Val(cm)	800 millibars D-Val(cm)
330	850mb D-Val(m)	850 millibars delta value
331	850mb D-Val(cm)	850 millibars D-Val(cm)
340	900mb D-Val(m)	900 millibars delta value
341	900mb D-Val(cm)	900 millibars D-Val(cm)
360	925mb D-Val(m)	925 millibars delta value
361	925mb D-Val(cm)	925 millibars D-Val(cm)
370	975mb D-Val(m)	975 millibars delta value
371	975mb D-Val(cm)	975 millibars D-Val(cm)
380	C/km ² * 10000	degrees Celsius per square kilometer x 10 to 4th
381	C/ft ² * 10000	degrees Celsius per square feet x 10 to 4th
390	C	degrees Celsius
400	F	degrees Fahrenheit
401	F / 100 ft	degrees F per 100 feet
410	K/10000km ²	"Kelvin per 10,000 square kilometers"
420	K/m**2	degrees Kelvin per square meter
430	K/m	degrees Kelvin per meter
440	K	Kelvin
441	K*m/s	Kelvin * meter per second
442	K/s	Kelvin per second
450	M_units	M units
451	M_units/m	M units per meter
460	N_units	refractive N units

Unit ID	Unit Abbreviation	Unit Name
470	Nt/m**2	newtons per square meter
480	Pa/s	pascals per second
490	Pa	pascals
491	hPa	hectoPascals
492	kPa	kiloPascals
493	ln(kPa_)	ln kiloPascals
500	W/m**2	watts per square meter
501	W/srm**2	watts per ster radian meter squared
510	_	undefined
520	cal/cm**2/day	calories per square centimeter per day
530	cal/cm**2/hr	calories per square centimeter per hour
540	cm/day	centimeters per day
550	cm/s	centimeters per second
560	cm	centimeters
570	dB	decibels
580	day	days
590	deg/10	tens of degrees
600	deg_+E	longitude degrees 0-360 eastward
610	deg_+N	latitude degrees -90 to 90 north positive
620	deg_+W	longitude degrees 0-360 westward
630	deg	degrees
640	dval_cm	atmospheric dvalue (in cm)
650	dynes/cm**2	dynes per square centimeter
660	fath	fathoms
670	fraction	fraction (range 0 through 1)
680	ft/s	feet per second
690	ft	feet
700	gpcm	geopotential centimeters
710	gpm	geopotential meters
711	pv	Potential Vorticity (10e-6km^2/kg)
720	hr	hours

Unit ID	Unit Abbreviation	Unit Name
730	in/10	10ths of an inch
740	in	inches
750	kg/kg	kilograms per kilogram
751	kg/kg/s	kilograms per kilogram per second
760	kg/m**2/s	kilograms per square meter per second
770	kg/m**2	kilograms per square meter
780	kg/m**3	kilograms per cubic meter
781	(kg/m**3)(m/s)	kilograms per cubic meter times meters/second
790	kg	kilograms
800	km/s	kilometers per second
810	km	kilometers
820	kt	nautical miles per hour
830	m**2/s**2	square meters per square seconds
840	m**2/s	square meters per second
850	m/day	meters per day
851	1/m	per meter
852	m/m	meter per meter
860	m/s	meters per second
861	m/s**2	meters per second squared
870	mb	millibars
880	microbars/s	microbars per second
890	microsec	microsecond
900	millisec	millisecond
910	min	minutes
920	mm/hr	millimeters per hour
930	mm	millimeter
939	hm	hectoMeters
940	m	meters
941	Tropopause Level D-Val(cm)	Tropopause Level D-Val(cm)
942	Tropopause Level D-Val(m)	Tropopause Level D-Val(m)
950	nm/day	nautical miles per day

Unit ID	Unit Abbreviation	Unit Name
960	nm	nautical miles
970	numeric	numeric
980	percent	per cent
990	ppt	parts per thousand
1000	prob	probability
1010	s	seconds
1020	yd	yards
1029	eta	Eta Vertical Coordinate (range)
1030	sigma	Sigma Vertical Coordinate (range 1.000-0.000)
1040	dyne	dynes
1050	g	grams
1060	lb	pounds
1070	lb/ft**2	pounds per square feet
1080	g/cm**2	grams per square centimeter
1090	lb/in**2	pounds per square inch
2000	kg/cm**2	kilograms per square centimeter
2010	b	bars
2020	mg	milligrams
2030	tenths	tenths of a unit
2040	hz	hertz
2050	Dobson	Dobson
2060	J	Joules
2061	J/kg	Joules per kilogram
2062	J/m2	Joules per meter squared
3500	zone	convergence zone usage unit
5000	*	wildcard

4.2.2 MDGRID Geophysical Parameters Table

The MDGRID Geophysical Parameters Table contains the list of supported GRIB parameter identifiers that are common for all centers. The programmer must reference this table when setting the IParameterId field in the MAGRIDQUERY, MAGRID2DQuery, MAGRID2DINGEST, and MAGRID3DINGEST structures.

Parameter ID	Name	Unit ID
1	Pressure	490
2	Pressure Reduced to MSL	490
3	Pressure Tendency	480
5	ICAO Standard Atmosphere Reference Height	450
6	Geopotential	830
7	Geopotential Height	710
8	Geometric Height	940
9	Standard deviation of height	940
10	Total Ozone	2050
11	Temperature	440
12	Virtual Temperature	440
13	Potential Temperature	440
14	Pseudo-adiabatic Potential Temperature	440
15	Maximum Temperature	440
16	Minimum Temperature	440
17	Dew Point Temperature	440
18	Dew Point Depression	440
19	Lapse Rate	430
20	Visibility	940
21	Radar Spectra(1)	510
22	Radar Spectra(2)	510
23	Radar Spectra(3)	510
24	Parcel lifted index (to 500hPa)	440
25	Temperature Anomaly	440

Parameter ID	Name	Unit ID
26	Pressure Anomaly	490
27	Geopotential Height Anomaly	710
28	Wave Spectra(1)	510
29	Wave Spectra(2)	510
30	Wave Spectra(3)	510
31	Wind Direction	630
32	Wind Speed	860
33	Wind U-Component	860
34	Wind V-Component	860
35	Stream Function	840
36	Velocity Potential	840
37	Montgomery Stream Function	830
38	Sigma Coord. Vertical Velocity	30
39	Pressure Vertical Velocity	480
40	Geometric Vertical Velocity	860
41	Absolute Vorticity	30
42	Absolute Divergence	30
43	Relative Vorticity	30
44	Relative divergence	30
45	Vertical U-Component Shear	30
46	Vertical V-Component Shear	30
47	Direction of Current	630
48	Speed of Current	860
49	Current U-Component	860
50	Current V-Component	860
51	Specific Humidity	750
52	Relative Humidity	980
53	Humidity Mixing Ratio	750
54	Perceptible Water	770
55	Vapor Pressure	490
56	Saturation Deficit	490

Parameter ID	Name	Unit ID
57	Evaporation	770
58	Cloud Ice	770
59	Precipitation Rate	760
60	Thunderstorm Probability	980
61	Total Precipitation	770
62	Large Scale Precipitation (non-conv.)	770
63	Convective Precipitation	770
64	Snowfall Rate Water Equivalent	760
65	Water Equiv. of Accum. Snow Depth	770
66	Snow Depth	940
67	Mixed Layer Depth	940
68	Transient Thermocline Depth	940
69	Main Thermocline Depth	940
70	Main Thermocline Anomaly	940
71	Total Cloud Cover	980
72	Convective Cloud Cover	980
73	Low Cloud Cover	980
74	Middle Cloud Cover	980
75	High Cloud Cover	980
76	Cloud Water	770
77	Best Lifted Index (to 500 hPa)	440
78	Convective Snow	770
79	Large Scale Snow	770
80	Water Temperature	440
81	Land-sea Mask	670
82	Deviation of Sea Level From Mean	940
83	Surface Roughness Length	940
84	Albedo	980
85	Soil Temperature	440
86	Soil Moisture Content	770
87	Vegetation	980

Parameter ID	Name	Unit ID
88	Salinity	990
89	Density	780
90	Water Runoff	770
91	Ice Concentration	670
92	Ice Thickness	940
93	Direction of Ice Drift	630
94	Speed of Ice Drift	860
95	U-Component of Ice Drift	860
96	V-Component of Ice Drift	860
97	Ice Growth Rate	860
98	Ice Divergence	30
99	Snow Melt	770
100	Significant Height of Wind Waves and Swell	940
101	Wind Waves Direction	630
102	Significant Height of Wind Waves	940
103	Mean Period of Wind Wave	1010
104	Swell Waves Direction	630
105	Significant height of swell Waves	940
106	Mean Period of Swell Waves	1010
107	Primary Wave Direction	630
108	Primary Wave Mean Period	1010
109	Secondary Wave Direction	630
110	Secondary Wave Mean Period	1010
111	Net Short-Wave Radiation (surface)	500
112	Net Long-Wave Radiation (surface)	500
113	Net Short-Wave Radiation (top of atmos.)	500
114	Net Long-Wave Radiation (top of atmos.)	500
115	Long-Wave Radiation	500
116	Short-Wave Radiation	500
117	Global-Wave Radiation	500
118	Brightness Temperature	440

Parameter ID	Name	Unit ID
119	Long-Wave Radiation	501
120	Short-Wave Radiation	501
121	Latent Heat Net Flux	500
122	Sensible Heat Net Flux	500
123	Boundary Layer Dissipation	500
124	Momentum Flux, U Component	470
125	Momentum Flux, V Component	470
126	Wind Mixing Energy	2060
127	Image data	510

The unit ID value maps back to the previously described MDGRID Units Table. This is the unit type of the ingest grid, prior to any conversions.

4.2.3 MDGRID Site-Specific Geophysical Parameters Table

The MDGRID Site-Specific Geophysical Parameters Table contains the list of supported GRIB parameter identifiers that are unique for a given center. The programmer must reference this table when setting the IParameterId field in the MAGRIDQUERY, MAGRID3DQUERY, MAGRID2DINGEST, and MAGRID3DINGEST structures.

Center ID	Sub Center ID	Parameter ID	Name	Unit ID
7	0	128	Mean Sea Level Pressure (Standard Atmosphere Reduction)	490
7	0	129	Mean Sea Level Pressure(MAPS System Reduction)	490
7	0	130	Mean Sea Level Pressure(ETA Model Reduction)	490
7	0	131	Surface lifted index	440
7	0	132	Best (4 layer) lifted index	440
7	0	133	K index	440
7	0	134	Sweat index	440
7	0	135	Horizontal moisture divergence	751
7	0	136	Vertical speed shear	30
7	0	137	3-hr pressure tendency Std. Atmos. Reduction	480
7	0	138	Brunt-Vaisala frequency (squared)	20
7	0	139	Potential vorticity (density weighted)	32
7	0	140	Categorical rain (yes=1; no=0)	5000
7	0	141	Categorical freezing rain (yes=1; no=0)	5000
7	0	142	Categorical ice pellets (yes=1; no=0)	5000
7	0	143	Categorical snow (yes=1; no=0)	5000
7	0	144	Volumetric soil moisture content	670
7	0	145	Potential evaporation rate	500
7	0	146	Cloud workfunction	2061

Center ID	Sub Center ID	Parameter ID	Name	Unit ID
7	0	147	Zonal flux of gravity wave stress	470
7	0	148	Meridional flux of gravity wave stress	470
7	0	149	Potential vorticity	711
7	0	150	Covariance between meridional & zonal components of wind	830
7	0	151	Covariance between temp & zonal component of wind	441
7	0	152	Covariance between temp & meridional component of wind	441
7	0	153	Cloud water	750
7	0	154	Ozone mixing ratio	750
7	0	155	Ground Heat Flux	500
7	0	156	Convective inhibition	2061
7	0	157	Convective Available Potential Energy	2061
7	0	158	Turbulent Kinetic Energy	2061
7	0	159	Condensation pressure of parcel lifted from surface	490
7	0	160	Clear Sky Upward Solar Flux	500
7	0	161	Clear Sky Downward Solar Flux	500
7	0	162	Clear Sky upward long wave flux	500
7	0	163	Clear Sky downward long wave flux	500
7	0	164	Cloud forcing net solar flux	500
7	0	165	Cloud forcing net long wave flux	500
7	0	166	Visible beam downward solar flux	500
7	0	167	Visible diffuse downward solar flux	500
7	0	168	Near IR beam downward solar flux	500
7	0	169	Near IR diffuse downward solar flux	500
7	0	172	Momentum flux	500

Center ID	Sub Center ID	Parameter ID	Name	Unit ID
7	0	173	Mass point model surface	5000
7	0	174	Velocity point model surface	5000
7	0	175	Model layer number (from bottom up)	5000
7	0	176	latitude (-90 to +90)	630
7	0	177	east longitude (0-360)	630
7	0	181	x-gradient of log pressure	851
7	0	182	y-gradient of log pressure	851
7	0	183	x-gradient of height	852
7	0	184	y-gradient of height	852
7	0	189	Virtual potential temperature	440
7	0	190	Storm relative helicity	830
7	0	191	Probability from ensemble	970
7	0	192	Probability from ensemble normalized to climate expectancy	970
7	0	193	Probability of precipitation	980
7	0	194	Probability of frozen precipitation	980
7	0	195	Probability of freezing precipitation	980
7	0	196	u-component of storm motion	860
7	0	197	v-component of storm motion	860
7	0	201	Ice-free water surface	980
7	0	204	downward short wave rad. flux	500
7	0	205	downward long wave rad. flux	500
7	0	206	Ultra violet index(1hr integration centered at solar noon)	2062
7	0	207	Moisture availability	980
7	0	208	Exchange coefficient	781
7	0	209	No. of mixed layers next to surface	970
7	0	211	upward short wave rad. flux	500

Center ID	Sub Center ID	Parameter ID	Name	Unit ID
7	0	212	upward long wave rad. flux	500
7	0	213	Amount of non-convective cloud	980
7	0	214	Convective Precipitation rate	760
7	0	215	Temperature tendency by all physics	442
7	0	216	Temperature tendency by all radiation	442
7	0	217	Temperature tendency by non-radiation physics	442
7	0	218	precip.index(0.0-1.00)(see note)	670
7	0	219	Std. dev. of IR T over 1x1 deg area	440
7	0	220	Natural log of surface pressure	493
7	0	221	Planetary boundary layer height	940
7	0	222	5-wave geopotential height	710
7	0	223	Plant canopy surface water	760
7	0	226	Blackadar's mixing length scale	940
7	0	227	Asymptotic mixing length scale	940
7	0	228	Potential evaporation	760
7	0	229	Snow phase-change heat flux	500
7	0	231	Convective cloud mass flux	480
7	0	232	Downward total radiation flux	500
7	0	233	Upward total radiation flux	500
7	0	234	Baseflow-groundwater runoff	760
7	0	235	Storm surface runoff	760
7	0	237	Total ozone	760
7	0	238	Snow cover	980
7	0	239	Snow temperature	440
7	0	241	Large scale condensat. heat rate	442
7	0	242	Deep convective heating rate	442
7	0	243	Deep convective moistening rate	751

Center ID	Sub Center ID	Parameter ID	Name	Unit ID
7	0	244	Shallow convective heating rate	442
7	0	245	Shallow convective moistening rate	751
7	0	246	Vertical diffusion heating rate	442
7	0	247	Vertical diffusion zonal acceleration	861
7	0	248	Vertical diffusion meridional accel	861
7	0	249	Vertical diffusion moistening rate	751
7	0	250	Solar radiative heating rate	442
7	0	251	long wave radiative heating rate	442
7	0	252	Drag coefficient	5000
7	0	253	Friction velocity	860
7	0	254	Richardson number	5000
7	0	255	Missing	490
57	10	129	Solar Flux	5010
58	0	128	Primary Layer Depth	690
58	0	129	Temperature at PLD	690
58	0	130	Temp Between PLD+ and PLD+ Layers	390
58	0	131	Temp Differences Between Levels	381
58	0	133	Ground Sea Temperature	440
58	0	139	Visual Sea Height	940
58	0	140	Short Wave Pattern	940
58	0	141	Long Wave Pattern	940
58	0	142	Ditch Headings	590
58	0	143	Wetness	510
58	0	144	Fog Probability	980
58	0	145	Freezing Level	940
58	0	153	Surface Stress U-Component	470
58	0	154	Surface Stress V-Component	470

Center ID	Sub Center ID	Parameter ID	Name	Unit ID
58	0	155	White Cap Probability	980
58	0	162	Temp at Depth	390
58	0	170	Evaporation Duct Height	940
58	0	171	Rain Mixing Ratio	750
58	0	172	Ice Mixing Ratio	750
58	0	173	Snow Mixing Ratio	750
58	0	174	Cloud Mixing Ratio	750
58	0	175	Water Vapor Mixing Ratio	750
58	0	176	Low Cloud Base Height	940
58	0	177	Low Cloud Top Height	940
58	0	178	Middle Cloud Base Height	940
58	0	179	Middle Cloud Top Height	940
58	0	180	Water Temperature	390
58	0	181	High Cloud Top Height	940
58	0	182	Cb Cloud Base Height	940
58	0	183	Cb Cloud top Height	940
58	0	185	Evaporation Duct Height	940
58	0	186	Frontal Analysis	420
58	0	191	Ice Pressure	470
58	0	201	Sea Surface Temperature	440
58	0	202	Mixed Layer Depth	940
58	0	203	Deep Sound Channel Axis	940
58	0	204	Shallow Sound Channel Axis	940
58	0	205	Sonic Layer Depth	940
58	0	206	Depth Excess (from the channel axis to the water bottom)	940
58	0	209	Visual Sea Height	690
58	0	211	Clean Air Turbulence Index	510

Center ID	Sub Center ID	Parameter ID	Name	Unit ID
58	0	212	Aircraft Icing Probability Index	510
58	0	213	Ditch Headings	590
58	0	214	Convergence Zone Probability Index	510
58	0	215	M-unit	450
58	0	217	Terrain Height	940
58	0	218	Ground Wetness	770
58	0	219	1000-500 Thickness	940
58	0	221	Sonic Layer Cutoff Frequency	30
58	0	222	Total Heat Flux	500
58	0	223	Shallow Sound Channel Intensity	430
58	0	224	Frontal Analysis	380
58	0	225	Surface Fog	510
58	0	230	Freezing Level Height	560
58	0	250	Modas Temperature	390
58	0	251	Modas Sound Velocity	860
58	0	252	Modas Salinity	990

4.2.4 MDGRID Production Centers Table

The MDGRID Production Centers Table contains the list of supported GRIB center identifiers. The preliminary release of the APIs contains the list of known production centers as described in the GRIB Specification Table 0. The Navy has not yet specified sub-center identifiers. When this occurs, updates will be made to this table. The programmer must reference this table when setting the lProductionCenterId field and the lSubCenterId field in the MAGRIDQUERY and MAGRIDSTORE structures. The MAGRIDRegister2DModel call may be used to dynamically update this table when a new center and model are available for ingest.

Center ID	Sub Center ID	Production Center Description
7	0	US Weather Service - National Met. Center
7	1	US National Center for NCEP Re-Analysis Project
7	2	US National Center for NCEP Ensemble Products
7	3	US National Center for NCEP Central Operations
7	4	US National Environmental Modeling Center
7	5	US National Hydrometeorological Prediction Center
7	6	US National Marine Prediction Center
7	7	US National Climate Prediction Center
7	8	US National Aviation Weather Center
7	9	US National Storm Prediction Center
7	10	US National Tropical Predication Center
8	0	US Weather Service - NWS Telecomms Gateway
9	0	US Weather Service - Field Stations
9	150	ABRFC - Arkansas-Red River RFC, Tulsa, OK
9	151	Alaska RFC Anchorage, AK
9	152	CBRFC - Colorado Basin RFC, Salt Lake City, UT
9	153	CNRFC - California-Nevada RFC, Sacramento, CA
9	154	LMRFC - Lower Mississippi RFC, Slidell, LA
9	155	MARFC - Middle Atlantic RFC, State College, PA
9	156	MBRFC - Missouri Basin RFC, Kansas City, MO

Center ID	Sub Center ID	Production Center Description
9	157	NCRFC - North Central RFC, Minneapolis, NM
9	158	NERFC - Northeast RFC, Hartford, CT
9	159	NWRFC - Northwest RFC, Portland, OR
9	160	OHRFC - Ohio Basin RFC, Cincinnati, OH
9	161	SERFC - Southeast RFC, Atlanta, GA
9	162	WGRFC - West Gulf RFC, Fort Worth, TX
9	170	OUN - Norman OK WFO
34	0	Japanese Meteorological Agency - Tokyo
52	0	National Hurricane Center, Miami
54	0	Canadian Meteorological Service - Montreal
57	0	US Air Force - Global Weather Center
57	10	US Air Force - TBD
58	0	US Navy - Fleet Numerical Oceanography Center
59	0	NOAA Forecast Systems Lab, Boulder CO
60	0	National Center for Atmospheric Research
74	0	UK Met Office - Bracknell
85	0	French Weather Service - Toulouse
97	0	European Space Agency
98	0	European Center for Medium-Range Weather Forecasts - Reading
99	0	DeBilt, Netherlands

4.2.5 MDGRID Models Table

The MDGRID Models Table contains the list of supported GRIB model types. This table gets queried when the API routine MAGRIDVerifyModel is used. The programmer must reference this table when setting the lGeneratingProcId field in the MAGRIDQUERY, MAGRID3DQUERY, MAGRID2DINGEST, and MAGRID3DINGEST structures for a given production center, subcenter, and grid id. Note that the MAGRIDRegisterModel may be called to dynamically update this table.

Center ID	Sub Center ID	Generating Process ID	Grid ID	Model Name
58	0	17	223	GSOWM
58	0	28	223	TOPS
58	0	40	223	EOTS
58	0	43	223	OTIS
58	0	46	240	NMC
58	0	48	240	ECMWF
58	0	58	223	NOGAPS
58	0	58	240	NOGAPS
58	0	61	223	STRATOI
58	0	65	223	UANVA
58	0	68	223	TYAN
58	0	70	242	NORAPS Asia
58	0	71	245	NORAPS CONUS
58	0	72	244	NORAPS Europe
58	0	73	243	NORAPS Indian Ocean
58	0	75	237	NORAPS Southern Cal
58	0	76	249	NORAPS Bosnia
58	0	82	223	TROPO
58	0	83	223	GOXM

4.2.6 MDGRID Registration Table

The MDGRID Registration Table contains the list of supported model registrations. The programmer must reference this table when setting the ulSize field in the MAGRID2DINGEST and MAGRID3DINGEST structures for a given production center, subcenter, and grid id. 2D Grids when produced by a given model (Center, sub-Center, Generating Process, and Grid ID) have a given X dimension size and Y dimension size; consequently, the size of the grid to be stored should always be MaxXpt * MaxYpt * sizeof(float). When produced by a given model (Center, sub-Center, Generating Process, and Grid ID), 3D grids have a given X dimension size, Y dimension size, and Z dimension. Consequently, the size of the grid to be stored should always be MaxXpt * MaxYpt * MaxZpt * sizeof (float). Note that the MAGRIDRegisterModel may be called to dynamically update this table with new geometry for new models.

Center ID	Sub-Center ID	Grid ID	AOI ID	Reg Lat	Reg Lon	RegX	RegY	X Distance	Y Distance	Max X pts	Max Y pts	Max Z pts	Projection	Scan mode
58	0	223	1	0.00	-120.0	73.0	37.0	277.842	277.842	144	73	0	16	3
58	0	237	24	29.0	-126.5	1.0	51.0	22.227	22.227	61	51	0	16	2
58	0	240	1	-90.0	0.00	1.0	181.0	111.137	111.137	360	181	0	16	2
58	0	242	23	4.0	92.0	1.0	105.0	55.569	55.569	133	105	0	16	2
58	0	243	26	0.0	28.0	1.0	81.0	55.569	55.569	149	81	0	16	2
58	0	244	25	27.0	-20.0	1.0	67.0	55.569	55.569	141	67	0	16	2
58	0	245	20	5.0	-124.0	1.0	93.0	55.569	55.569	145	93	0	16	2
58	0	249	21	32.5	5.0	1.0	76.0	22.227	22.227	101	76	0	16	2

4.2.7 MDGRID Area of Interest Table

The MDGRID AOI Table contains the list of AOIs for supported models. This table is provided only for reference purposes. Note that the Registration and Model tables also contain the AOIID value. The programmer could use this to determine the AOI covered by the model (will need to cross-reference with the registration table). Note that MAGRIDRegisterModel may dynamically update this table.

AOI ID	AOI Name	North Lat	West Lon	South Lat	East Lon
1	GLOBAL	90.0	-180.0	-90.0	180.0
24	NORAPS Southern Cal	39.0	-126.50	29.00	-114.50
21	NORAPS Bosnia	47.50	5.0	32.50	25.0
23	NNORAPS Asia	56.0	92.0	4.0	158.0
20	NNORAPS CONUS	51.0	-124.0	5.0	-52.0
25	NNORAPS Europe	60.0	-20.0	27.00	50.00
26	NNORAPS Indian Ocean	40.0	28.00	0.0	102.0

4.3 Building Applications Using the MAGRID Libraries

This section contains four makefiles that can be used to build the MAGRID APIs. Static and dynamic makefiles are provided for the HP-UX environment and the Windows NT environment. The dynamic makefiles use the run-time libraries (*.sl) on HP-UX or (*.dll) on Windows NT to build the APIs. The static makefiles use the static libraries (*.a) on HP-UX, (*.lib) on Windows NT) to build the APIs.

4.3.1 Makefile Using the Dynamic/Static MAGRID Libraries on HP-UX

```
### Dynamically Linked Program
PROGRAM1 = MAGRID_Ingest_d

### Statically Linked Program
PROGRAM2 = MAGRID_Ingest_l

CFILE = main.c

OFILE = $(CFILE:.c=.o)

RM = /bin/rm -f
INFORMIXDIR = /opt/informix

INFORMIXLIBPATHS = -L$(INFORMIXDIR)/lib -L$(INFORMIXDIR)/lib/esql

INFORMIXLIBS = -liysql -lixgen -lixos -lixasf -lixgls \
    $(INFORMIXDIR)/lib/esql/checkapi.o -lnsl_s -lm -lv3 -lcl -lse

LIBS = -lMAGRIDAPI \
    -lMAGRIDKernel \
    -lMAGRIDutils

INCLUDES = -I$(MAGRID_HOME)/include

all : $(PROGRAM)

CFLAGS = $(INCLUDES) -Aa

# Dynamic Linked Program
$(PROGRAM1) : $(OFILE)
    $(CC) $(OFILE) \
        -L$(MAGRID_HOME)/bin $(LIBS) \
        $(INFORMIXLIBPATHS) $(INFORMIXLIBS) -lc -lm -o $(PROGRAM1)

# Dynamic Linked Program
$(PROGRAM2) : $(OFILE)
    $(CC) $(OFILE) \
        -L$(MAGRID_HOME)/lib $(LIBS) \
        $(INFORMIXLIBPATHS) $(INFORMIXLIBS) -lc -lm -o $(PROGRAM2)

clean :
    $(RM) $(OFILE) $(PROGRAM1) $(PROGRAM2) core
```

4.3.2 Makefile Using the Dynamic MAGRID Libraries on Windows NT

```
PROGRAM = MAGRID_Store  
  
CFILE = main.c  
  
OFILE = $(CFILE:.c=.obj)  
  
MAGRID_HOME = ..\..\..\..\..  
RM = -@del /Q /F /S  
LD = link  
  
INFORMIXLIB = $(INFORMIXDIR)\lib\isqlt07c.lib  
  
LIBS = $(MAGRID_HOME)\bin\MGRIDAPI.lib \  
       $(MAGRID_HOME)\bin\MGRIDKernel.lib \  
       $(MAGRID_HOME)\bin\MGRIDUtils.lib  
  
INCLUDES = -I$(MAGRID_HOME)\include  
  
all : $(PROGRAM)  
  
CFLAGS = $(INCLUDES) -MD  
  
$(PROGRAM) : $(OFILE)  
           $(LD) $(OFILE) /NODEFAULTLIB:LIBCMT.LIB /NODEFAULTLIB:LIBC.LIB\  
           $(LIBS) $(INFORMIXLIB) -OUT:$@  
  
clean :  
       $(RM) $(OFILE) $(PROGRAM)
```

4.3.3 Makefile Using the Static MAGRID Libraries on Windows NT

```
PROGRAM = MAGRID_Store  
  
CFILE = main.c  
  
OFILE = $(CFILE:.c=.obj)  
  
MAGRID_HOME = ..\..\..\..\..  
RM = -@del /Q /F /S  
LD = link  
  
INFORMIXLIB = $(INFORMIXDIR)\lib\isqlt07c.lib  
  
LIBS = $(MAGRID_HOME)\lib\MGRIDAPI.lib \  
       $(MAGRID_HOME)\lib\MGRIDKernel.lib \  
       $(MAGRID_HOME)\lib\MGRIDUtils.lib  
  
INCLUDES = -I$(MAGRID_HOME)\include  
  
all : $(PROGRAM)  
  
CFLAGS = $(INCLUDES)  
  
$(PROGRAM) : $(OFILE)  
           $(LD) $(OFILE) /NODEFAULTLIB:LIBC.LIB /NODEFAULTLIB:LIBCMT.LIB\  
           $(LIBS) $(INFORMIXLIB) -OUT:$@  
  
clean :  
        $(RM) $(OFILE) $(PROGRAM)
```

(This page intentionally left blank.)

5 CUSTOMIZING SEGMENTS

This section is tailored out.

(This page intentionally left blank.)

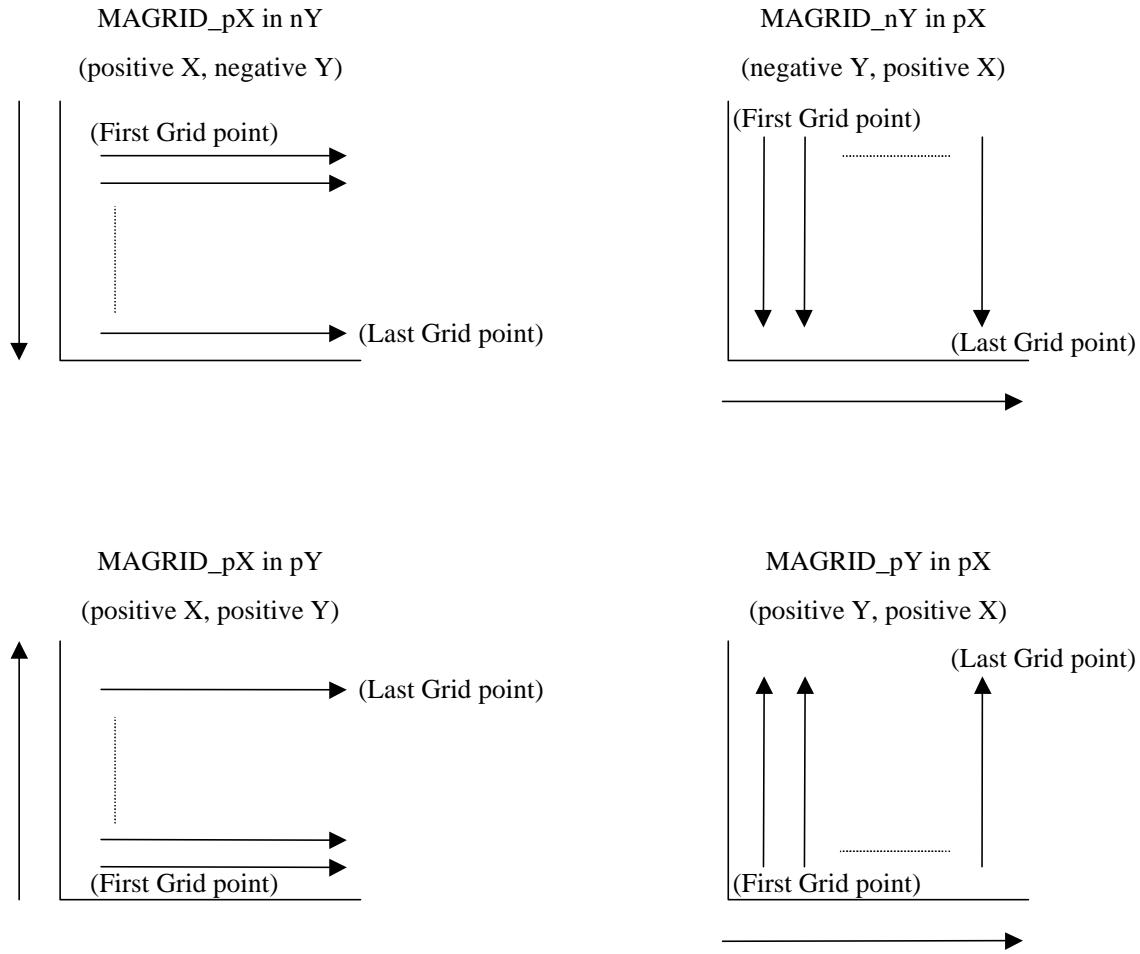
6 NOTES

6.1 Glossary of Acronyms

AESS	Allied Environmental Support System
AOI	Area of Interest
API	Application Program Interface
APIRM	API Reference Manual
BLOB	Binary Large Object
COE	Common Operating Environment
DII	Defense Information Infrastructure
EOF	End of File
GCCS	Global Command and Control System
IC4ISR	Integrated Command, Control, Communications, Computer, and Intelligence Surveillance Reconnaissance
IMOSS	Interim Mobile Oceanographic Support System
JMCIS	Joint Maritime Command Information System
JMS	Joint METOC Segment
LLT	Latitude-Longitude-Time
MAGRID	Grid Field API Segment of the TESS(NC) METOC Database
MDGRID	Grid Field Database Segment of the TESS(NC) METOC Database
METOC	Meteorological and Oceanographic

MIDDS	Meteorological Integrated Data Display System
NITES	Navy Integrated Tactical Environmental Subsystem
PC	Personal Computer
PM	Programming Manual
PS	Performance Specification
RDBMS	Relational Database Management System
SPAWAR	Space and Naval Warfare Systems Command
SQL	Structured Query Language
TESS(NC)	Tactical Environmental Support System Next Century

Appendix A - Scan Mode Values



PRINTED COPY IS UNCONTROLLED AND MAY BE OBSOLETE

ipd4600magridpmTES-10

(This page intentionally left blank.)